MSX BASIC

Albert Sickler

PHILIPS

Errata

S. 105: Fußnote:
*Hinweis: Wenn man wirklich unvorhersehbare Ergebnisse erzielen will, geht man aus von: RND(-TIME)

S. 136: In der Mitte der Seite:
Den Notenwert geben wir mit dem L-Code an. Ohne genauere Kennzeichnung geht der Computer aus von L4, und dies entspricht einer Viertelnote.

S. 156: Ende des ersten Absatzes:
Es ist zu berücksichtigen, daß der Computer ohne CLEAR nur 200 Bytes zur Speicherung von Buchstaben und dergleichen reserviert.

S. 157: Unter den Beispielen: COLOR,,5 wird COLOR,5.

S. 165: Der Text unter **KEY ON/KEY OFF** gehört ab dem Satz, der mit 'Man kann eventuell...' beginnt, unter die Beschreibung von
KEY (<n>) ON/KEY (<n>) OFF/KEY (<n>) STOP

S. 168: Erster Absatz: <255 wird >255
Außerdem müssen die Statements **ON SPRITE GOSUB...** und **ON STOP GOSUB...** verändert werden in **ON SPRITE GOSUB <Zeilennummer>** und **ON STOP GOSUB <Zeilennummer>**

S. 170: Zeile 20 im Programm wird: `20 SCREEN 2`

S. 173: Zweite Zeile des zweiten Programmbeispiels wird: `1.23E+02`

S. 178: Unter **WIDTH (<Anzahl Spalten>)**: 40 Spalten wird 37 Spalten.

S. 180/181: Text unter **INP (<Schnittstellennummer>)**: Liest ein Byte aus der angegebenen Schnittstellennummer. Für die Datenausgabe wird OUT verwendet (siehe OUT).

S. 182: Tabelle unten ändern in:

0 transparent
1 schwarz
2 bis 14 Farben bei Verwendung eines Farbfernsehers
15 weiß

S. 202: Zweiter Absatz: Der Satz 'LSET dient immer zur Speicherung von Strings...' wird ersetzt durch 'siehe Seite 205'.

S. 203: Unter 'Übersicht der Befehle':
[Verschiebung] nach **BLOAD"<Dateispezifizierung>"[,S]** entfält.

MSX BASIC

Philips-Ausgabe

Albert Sickler

MSX BASIC

Philips-Ausgabe

Kluwer Technische Boeken b.v.
Deventer – Antwerpen

Umschlag und Illustrationen: Wim Niessink

Übersetzung: Expertrans Zoetermeer, Holland.

© 1985 Kluwer Technische Boeken B.V. - Deventer

1. Druck 1985

Nichts aus dieser Ausgabe darf ohne vorherige schriftliche Genehmigung des Verlegers durch Druck, Fotokopie, Mikrofilm oder auf irgendwelche andere Weise vervielfältigt und/oder veröffentlicht werden.

Trotz aller dem Text gewidmeten Sorgfalt kann weder die Redaktion noch der Verleger für eventuellen Schaden haftbar gemacht werden, der sich aus irgendeinem in dieser Ausgabe vorkommenden Fehler ergeben sollte.

Inhalt

1 Der Computer: eine Einführung ... 9
2 MSX: ein Standard ... 17
3 Die ersten Schritte: über PRINT und rechnerische Bearbeitungen ... 21
4 Und jetzt ein BASIC-Programm ... 25
5 RUN, LIST, NEW, AUTO, RENUM und DELETE ... 31
6 Variablen und noch mehr über Rechnen ... 39
7 INPUT, READ und DATA ... 47
8 Zahlen, groß und klein und in allerlei Gestalt ...53
9 PRINT, TAB, LOCATE, PRINT USING und REM ... 61
10 Die Steueranweisungen: GOTO, IF...THEN, FOR...NEXT und ON...GOTO ... 67
11 Array (Auflistung) ... 79
12 Strings... spielen mit Buchstaben ... 85
13 Intermezzo: Können Computer denken? ... 95
14 Subroutinen (Unterprogramme) und mehr über das Entwikkeln eines Programms ... 101
15 Grafische Darstellungen: SCREEN, PSET, COLOR und PRESET ... 111
16 Grafische Darstellungen: LINE, DRAW, CIRCLE und PAINT ... 119
17 Intermezzo: über einst und jetzt ... 129
18 Ton: BEEP, PLAY und SOUND ... 133
19 Sprites ... 143

Anhang A: Dateien ... 148
Anhang B: Zeichen und ihre Kodes ... 151
Anhang C: Speichereinteilung ... 153
Anhang D: Übersicht der BASIC-Anweisungen und -Befehle ... 154
Anhang E: Übersicht der BASIC-Funktionen und Systemvariablen ... 179
Anhang F: Besondere Zeichen und Ausdrücke ... 186
Anhang G: Fehlermeldungen ... 188
Anhang H: Tastatur ... 192
Anhang I: Steuerkodes und Farbkodes ... 193

Anhang J: MSX Disk-BASIC ...195
Anhang K: MSX Quick Disk BASIC ...209
Anhang L: MSX RS232C-Schnittstelle ... 215
Anhang M: Aufbau der Symbole ...224
Anhang N: Reservierte Namen ... 228
Anhang O: Technische Information ... 229

Programme ... 232

Vorwort

Hinter den drei Buchstaben MSX verbirgt sich eine ganze Welt. Diese Welt wird bestimmt durch eine Vielzahl von Computermarken, die eher durch ihre Übereinstimmung als durch Unterschiedlichkeit auffallen. Zum erstenmal wird in der Welt der Heimcomputer ein Standard eingeführt, und hieraus werden sich ungezweifelt enorme Konsequenzen ergeben.

Mit der Einführung von MSX-Computern für das große Publikum wird sich mit Sicherheit ein großes Interesse ergeben, was nun eigentlich MSX-Computer sind und was man mit MSX-BASIC machen kann.

Bei der Auswahl eines MSX-Computers haben Sie sich schließlich für einen Philips-Computer entschieden. Das war die richtige Wahl, denn jedermann weiß, wie dieser Elektronik-Gigant im Qualitätsbereich seine Sporen verdient hat.

Für viele dürfte dies eine erste Bekanntschaft sein, und bei dieser Bekanntschaft erwartet der Leser gewiß Antwort auf die Frage, wie er die ersten Schwellen überschreiten kann. Für diese Leser ist dieses Buch geschrieben.

Albert Sickler

Dieses Kapitel gibt eine kurze Einführung über den Computer selbst. Wir lernen die Begriffe Programm, zentrale Verarbeitungseinheit, RAM- und ROM-Speicher sowie externe Speicher kennen.

Ein Wörterverzeichnis vermittelt häufig vorkommende Begriffe aus der Welt der Computer. Hier tritt u.a. der Begriff Operating System hervor, und dieser Begriff bezieht sich auf eine Sammlung von Programmen, die zu einem wesentlichen Teil die Anwendungsmöglichkeiten des Computers bestimmen.

1. Der Computer: eine Einführung

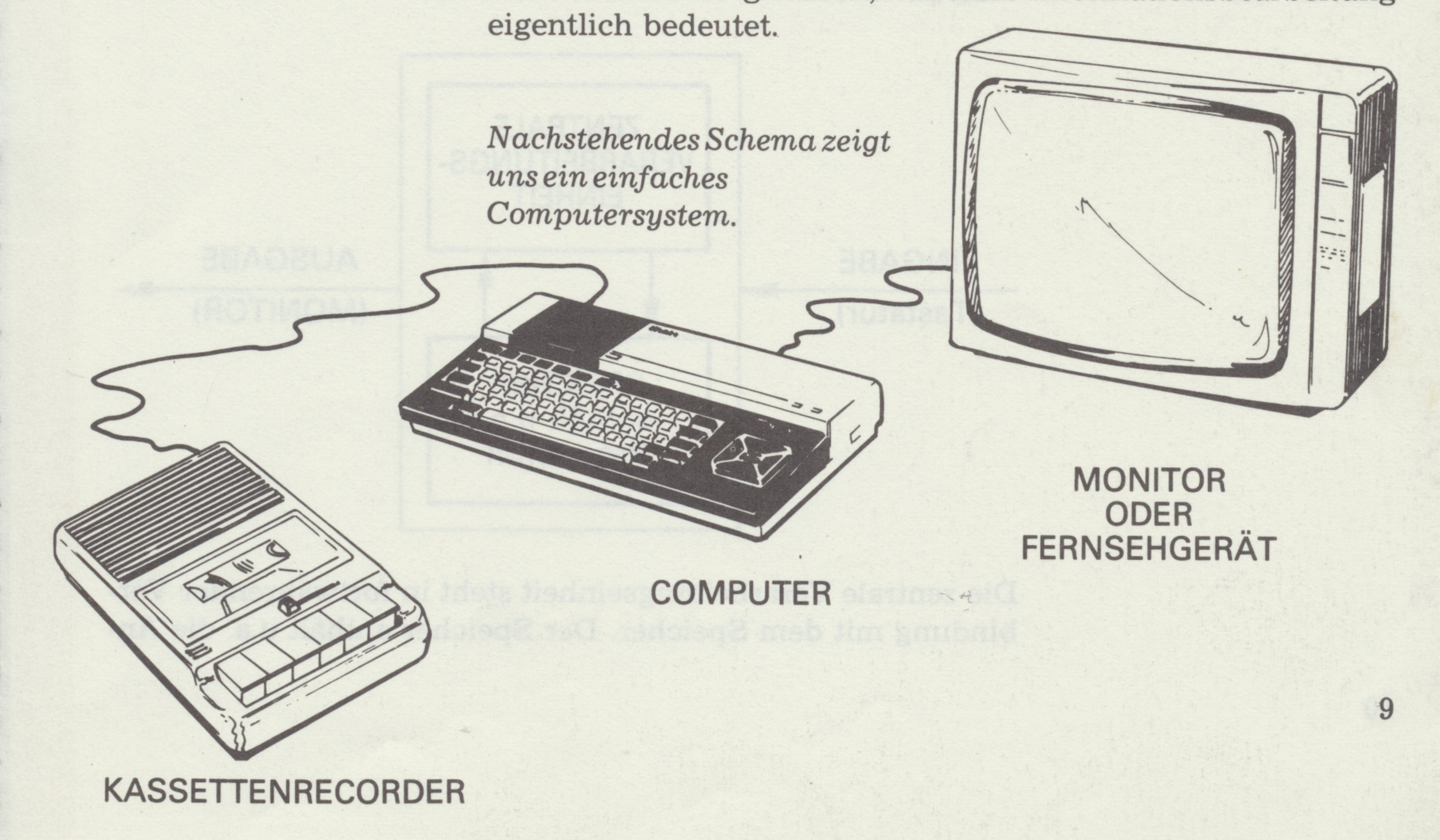
Einführung

Dieses Buch hat zwar vor allem die Aufgabe, den Anfänger zu unterrichten, wie er mit MSX-BASIC arbeiten kann, doch dazu muß man wohl erst einmal etwas über den Computer selbst erzählen.

MSX-BASIC ist schließlich nichts anderes als ein Ausdrucksmittel, mit dem wir dem Computer sagen, was getan werden muß. Daher ist es wichtig zu wissen, was ein Computer eigentlich ist.

Der Computer sieht beim ersten Anblick aus wie ein Kasten, an dem sich eine Tastatur befindet. Wenn wir den Computer mit irgendeinem anderen Apparat vergleichen, fällt vor allem die Übereinstimmung mit einer Schreibmaschine auf: auch bei der Schreibmaschine dominiert die Tastatur. Der Unterschied zwischen Schreibmaschine und Computer ist jedoch, daß der Computer mit einer großen Anzahl elektronischer Schaltungen versehen ist, die zur 'Informationsbearbeitung' notwendig ist. Wir werden in der Folge sehen, was diese Informationsbearbeitung eigentlich bedeutet.

Nachstehendes Schema zeigt uns ein einfaches Computersystem.



Auf vorstehender Abbildung sehen wir, daß unser Computer zugleich mit einem Bildschirm und einem Kassettenrecorder verbunden ist. Bei unserem MSX- Computer können wir von einem Fernsehgerät Gebrauch machen. Es zeigt uns die Daten, die der Computer produziert. Computerfachleute sprechen in diesem Fall von einem 'Ausgabegerät'.

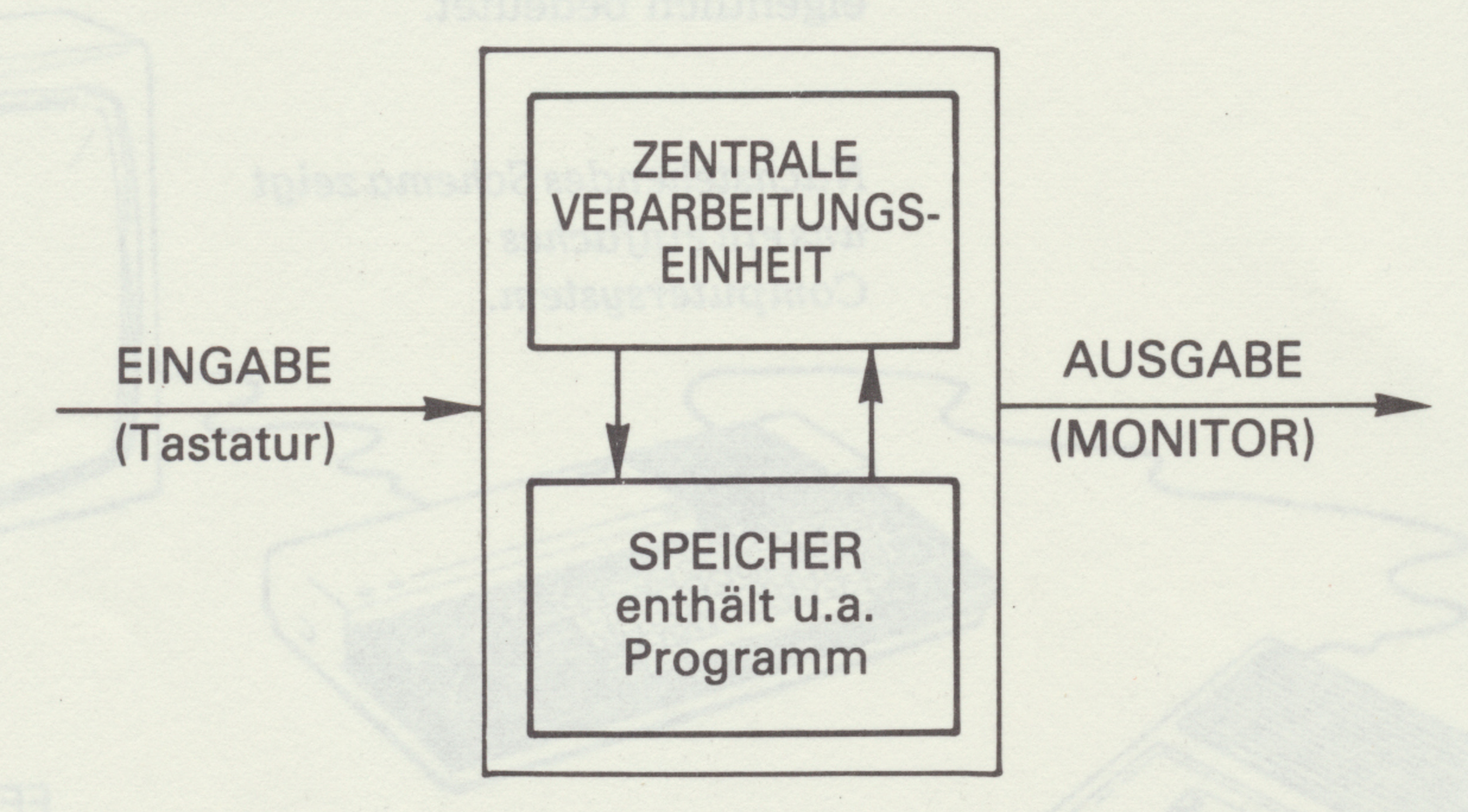
In der einfachsten Situation geben wir auf der Tastatur ein Programm ein. Ein Programm besteht aus einer Folge von Anweisungen, die dem Computer genau angeben, was er zu tun hat.

Für den Anfänger klingt dies alles etwas schwierig. An Beispielen werden wir noch sehen, wie einfach dies in der Praxis geschieht. Ein von uns eingetipptes Programm wird in einem elektronischen Speicher aufbewahrt. Der elektronische Speicher ist eine der elektronischen Schaltungen, die unser Computer enthält.

Erst nach dem Eintippen eines besonderen Wortes führt der Computer das eingetippte Programm dann auch wirklich aus. Solch ein spezielles Wort, mit dem wir deutlich machen, daß der Computer eine besondere Aktion vornehmen muß, beispielsweise das wirkliche Ausführen eines im Speicher enthaltenen Programmes, bezeichnen wir als Befehl.

Der Prozessor

Um die im Speicher enthaltenen Anweisungen ausführen zu können, hat der Computer noch eine besondere elektronische Schaltung, die wir als zentrale Verarbeitungseinheit bezeichnen.
Folgendes Schema verdeutlicht dies alles:



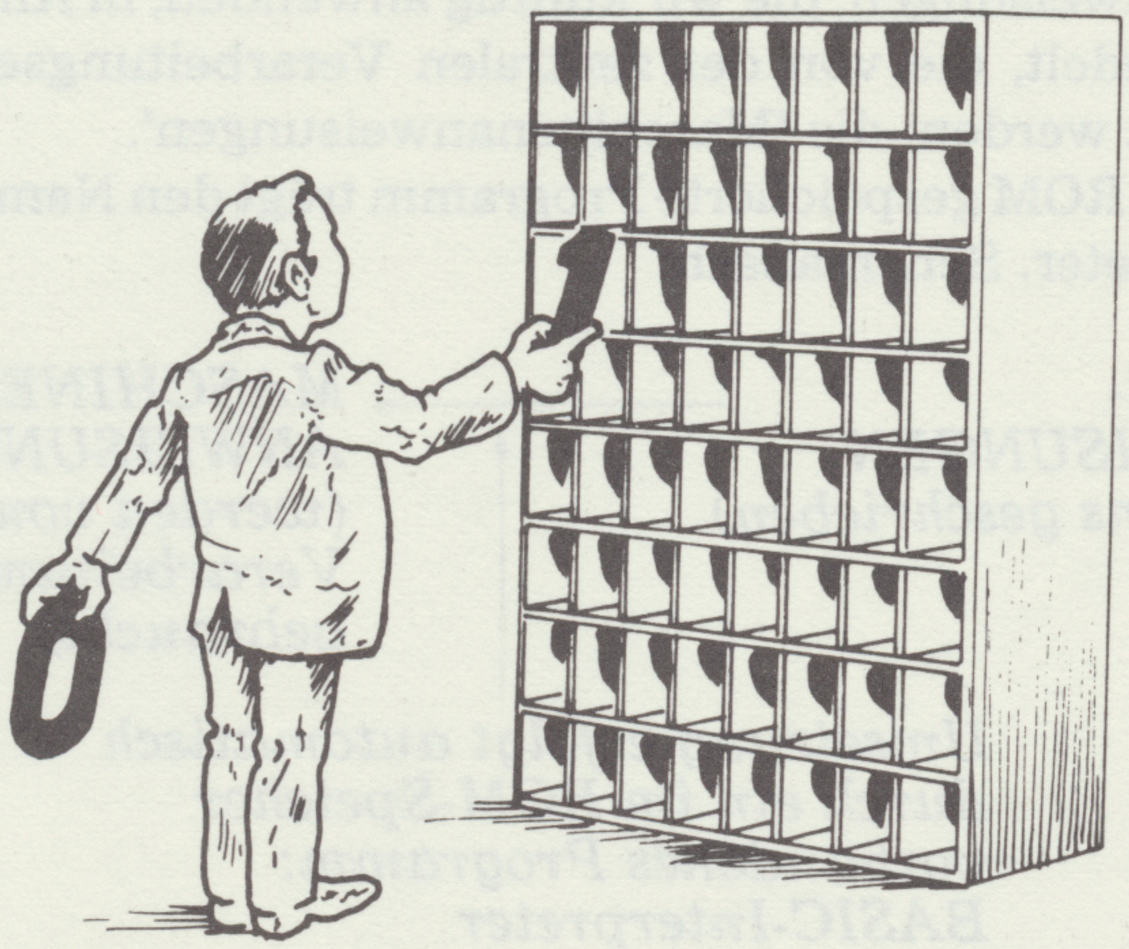
Die zentrale Verarbeitungseinheit steht in fortwährender Verbindung mit dem Speicher. Der Speicher enthält u.a. die An-

weisungen, die auszuführen sind. Bei der Ausführung von Berechnungen werden außerdem auch Zwischenergebnisse in den Speicher aufgenommen. Man kann hierbei etwa an einen Schmierzettel denken:
auch darauf kann man bei komplizierten Berechnungen die Zwischenergebnisse notieren.
Die Sprache der Computerleute ist ziemlich stark mit amerikanischen Ausdrücken gespickt. Das ist gar nicht erstaunlich, denn viele Entwicklungen im Computerbereich haben schließlich in Amerika angefangen.

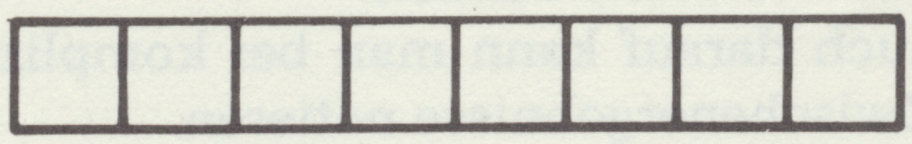
Einer der Ausdrücke, die man oft hören kann, ist das Wort 'Processor', und das ist der englische Ausdruck für eine Verarbeitungseinheit. Wenn wir also über den Prozessor eines Computers sprechen, handelt es sich um die elektronischen Schaltungen, in denen die faktischen Bearbeitungen stattfinden. Bei unserem MSX-Computer ist dieser Prozessor eine sehr umfassende Schaltung, die insgesamt auf einem Siliziumscheibchen untergebracht ist. Solch eine Kompaktschaltung auf einem Siliziumscheibchen bezeichnen wir als einen Chip. Unser Prozessor in Form eines Chips hat sogar einen besonderen Namen: wir bezeichnen ihn als einen Microprozessor - eben weil diese Schaltung so winzig klein ist.
Es gibt nur wenige Hersteller von Microprozessoren. Einer der bekanntesten Typen unter den Microprozessoren hat den etwas geheimnisvollen Namen Z80. Dies ist der Microprozessor, der in allen MSX-Computern verwendet wird.

Der Speicher: RAM und ROM

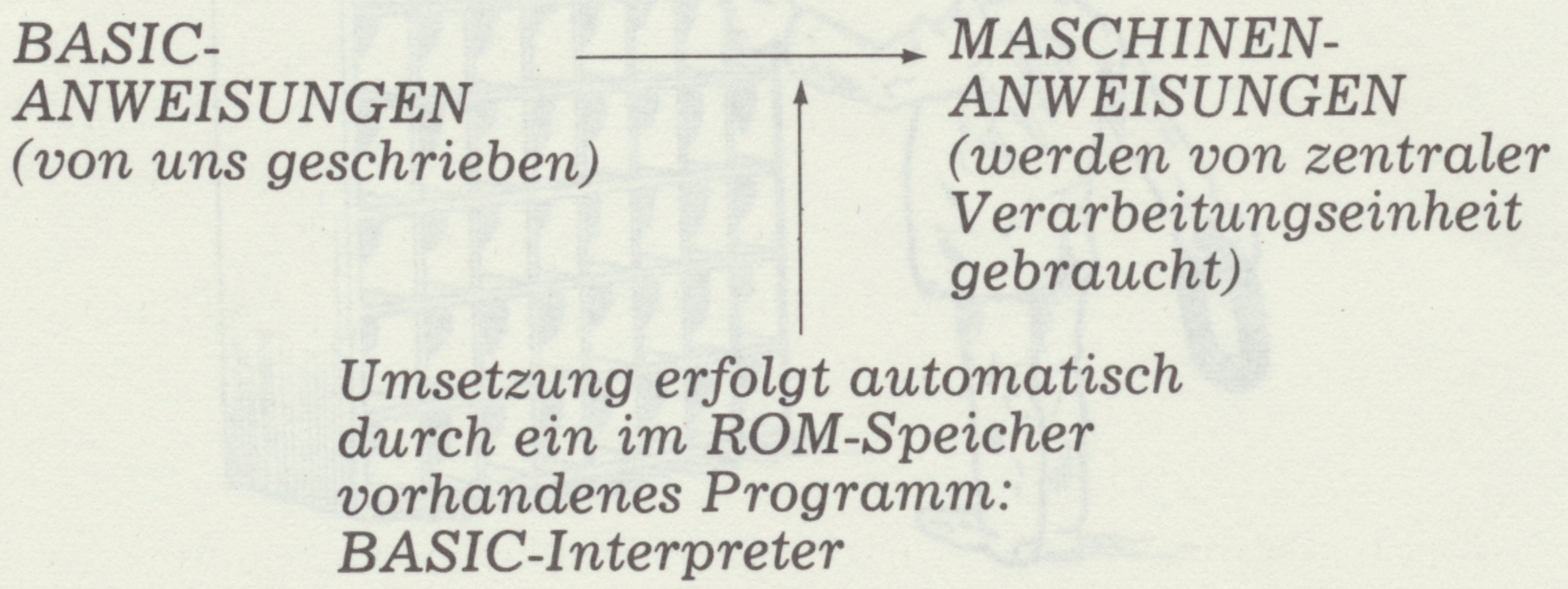
Zur ordentlichen Unterbringung der Daten ist der elektronische Speicher wie ein großes Ladenregal angeordnet.



Jede der Laden ist in eine Reihe von Fächern unterteilt. Bei den meisten Heimcomputern, auch bei unserem MSX, wird eine Aufteilung von 8 Fächern angewendet:



In jedem der Fächer können wir nur eine 1 oder eine 0 speichern. Wir sagen daher auch: jedes Fach hat eine Speicherkapazität von 1 Bit.
Oft begegnet man auch dem Wort 'Byte'. Ein Byte entspricht einer Lade von 8 Fächern, also kurzum einer Speicherkapazität von 8 Bits.
Computerleute sprechen in der Regel nicht von Laden, sondern von Wörtern. Wenn ein Wort also 8 Fächer enthält, hat es eine Speicherkapazität von 8 Bits, also von 1 Byte. Computerspeicher enthalten Tausende solcher Wörter. So wie wir 1000 Meter abkürzen mit 1 km, sprechen wir bei einem Speicher von beispielsweise 16 000 Bytes von 16 Kbyte.
MSX-Computer haben auf jeden Fall eine Speicherkapazität von 8 Kbyte. Doch das ist nur die Mindestmenge, und dies bedeutet wiederum, daß der MSX- Computer in der Praxis eine viel größere Speicherkapazität hat.
Die Speicherschaltung unseres Computers ist so gemacht, daß wir jede beliebige Lade zum Eingeben wie zum Ausgeben benutzen können. In diesem Fall sprechen wir von einem 'willkürlich zugänglichen Speicher' (englisch: Random Access Memory, abgekürzt RAM).
Außer dem RAM-Speicher gibt es bei Computern auch noch ROM-Speicher. Darin hat der Hersteller Fertigprogramme eingespeichert. ROM bedeutet 'Read Only Memory', und dies bedeutet 'Speicher nur zum Lesen'.
So enthält der ROM-Speicher u.a. ein Programm, das die BASIC-Anweisungen, die wir künftig anwenden, in Anweisungen umwandelt, die von der zentralen Verarbeitungseinheit verwendet werden: die 'Maschinenanweisungen'.
Das im ROM gespeicherte Programm trägt den Namen BASIC-Interpreter. Schematisch:



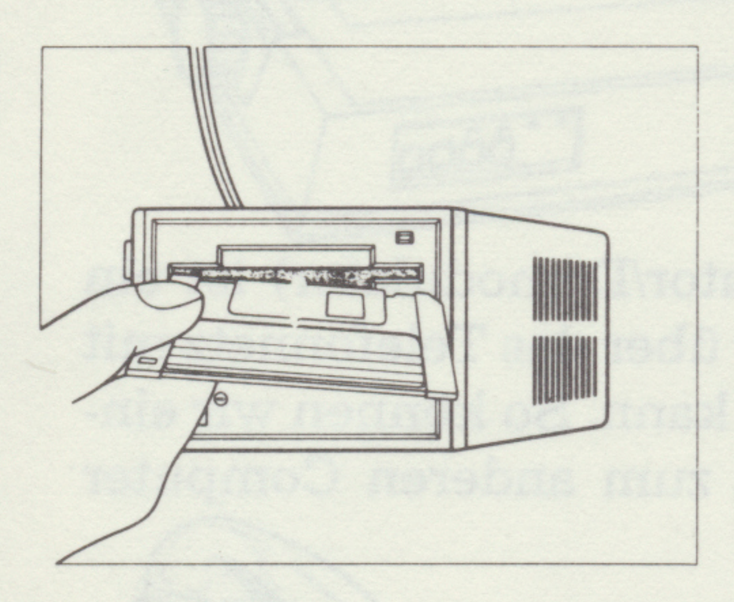
MSX-Computer haben als Standard einen ROM-Speicher von 32 Kbyte.

Lauter Einsen und Nullen

Wir kommen noch einmal auf den Speicher zurück, in dem offenbar nur Einser und Nullen gespeichert werden können. Aber wenn Sie dann auf Ihre Tastatur schauen, denken Sie wahrscheinlich: Was soll denn das? Auf Ihrer Tastatur sind schließlich nur allerlei Zeichen zu sehen, und nicht nur die 1 und die 0. Doch das ganze Geheimnis ist eben, daß alle Buchstaben und Zahlen, die wir einspeichern, in Einsen und Nullen umgesetzt werden. Das Morsealphabet zeigt beispielsweise, wie man Buchstaben auf diese Weise umsetzen kann.
Denken Sie daran, daß die Tastatur mit einer komplizierten Schaltung verbunden ist, wodurch jede Taste, die wir drücken, sofort eine typische Reihe von Einsen und Nullen verursacht.
Auch die Ausgabe des Computers besteht im Prinzip aus Einsen und Nullen. Glücklicherweise gibt es wiederum elektronische Schaltungen, die diese Reihen als lesbare Buchstaben oder sonstige Zeichen auf dem Bildschirm sichtbar werden lassen.

Externe Speicher

Auf der ersten Zeichnung sehen wir außerdem noch einen Kassettenrecorder. Bei MSX-Computern können wir tatsächlich einen gewöhnlichen oder etwas angepaßten Kassettenrecorder verwenden. Man kann für solch einen Kassettenrecorder Kassetten kaufen, auf denen bereits Programme festgelegt sind, genau wie wir auch Musikkassetten kaufen können. Durch einen besonderen Befehl können wir den Computer beauftragen, das auf der Kassette festgelegte Programm 'einzulesen', d.h. es in den Speicher des Computers aufzunehmen. Erst wenn es darin eingespeichert ist, kann es auch wirklich ausgeführt werden.
Außerdem kann man auch selbst entwickelte Programme auf der Kassette registrieren.
Begeisterte Programmierer haben oft eine umfangreiche Kassettensammlung, auf denen sie ihre schöpferische Leistungen festgehalten haben.



Bei MSX-Computern kann man auch sogenannte Floppy Disks oder Disketten verwenden. Deren Funktion ist vergleichbar mit der einer Kassette. Auf einem Floppy können wir jedoch sehr viel schneller die Daten (z.B. ein Programm) vom Computer aufsuchen lassen, als dies bei einem Kassettenrecorder möglich ist.

Kleines Wörterbuch

Wir wollen anschließend noch einige häufig gebrauchte Ausdrücke erklären.

Hardware: Unter Hardware versteht man alles, was bei einem Computer greifbar ist oder, volkstümlich gesagt, kaputtgehen kann. Gehäuse, elektronische Schaltungen usw. gehören zur Hardware.

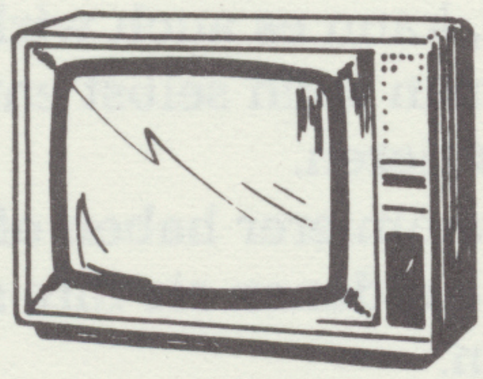
Software: Dieser Ausdruck wird im allgemeinen dann gebraucht, wenn über 'Programme' gesprochen wird.

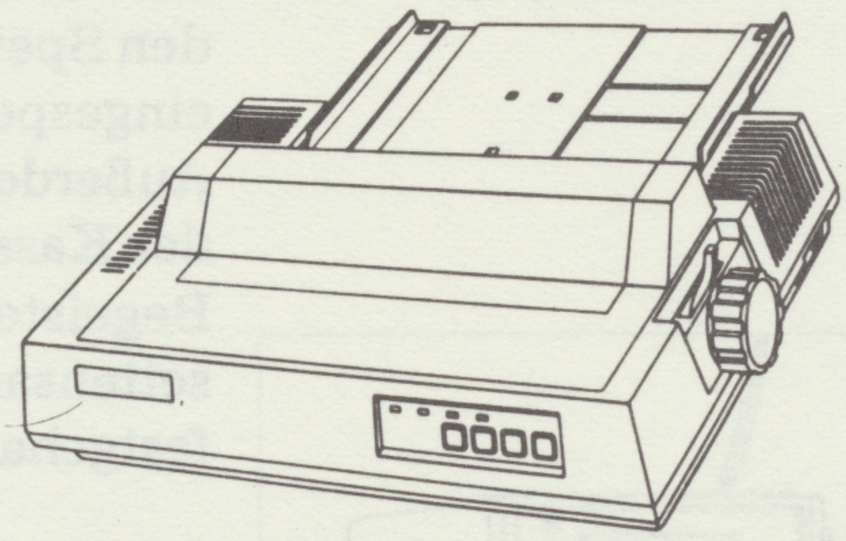
Microprozessor: Die zentrale Verarbeitungseinheit besteht aus einem Chip (Kompaktschaltung auf einem Siliziumscheibchen). Es gibt nur eine beschränkte Typenzahl von Microprozessoren, z.B. Z80, 6502 und 6809. (Man kann diese Bezeichnungen mit den Automarken vergleichen.) Die Maschineninstruktionen der genannten Microprozessoren sind unterschiedlich.
Bei MSX-Computern wird der Z80 Microprozessor verwendet.

Eingabegeräte: Alle an den Computer angeschlossenen Geräte zur Eingabe von Daten.

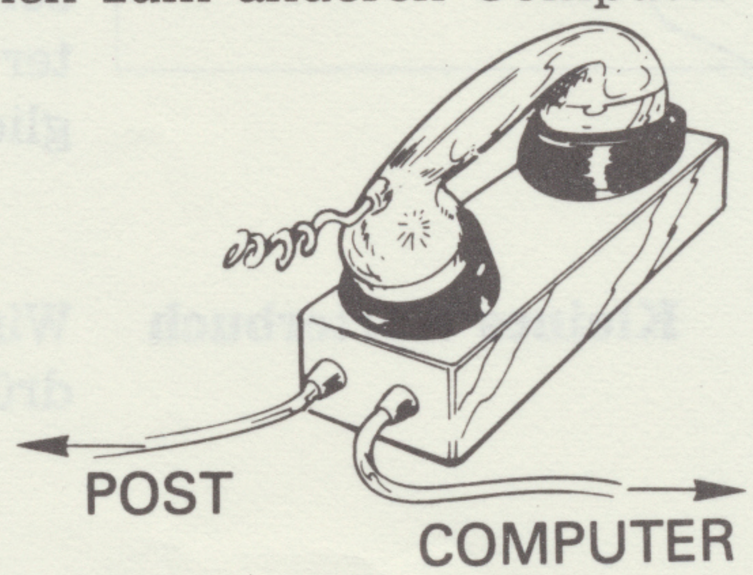
Ausgabegeräte: Alle an den Computer angeschlossenen Geräte zur Ausgabe der Daten. Bekannte Beispiele sind Fernseher und Monitor. Ein Monitor ist ein Bildschirm, der schärfere Bilder zeigt, als ein gewöhnliches Fernsehgerät.

Drucker: Der Drucker ist ein Gerät, mit dem Daten auf Papier ausgedruckt werden. Bekanntester Typ ist ein Matrixdrucker, von dem jedes Symbol in einer Serie Punkte wiedergegeben wird.





Modem: Ein Modem (Abkürzung für Modulator/Demodulator) ist ein Gerät, mit dem man einen Computer über das Telefonnetz mit einem anderen Computer verbinden kann. So können wir einfach ein Programm von dem einen zum anderen Computer übertragen.

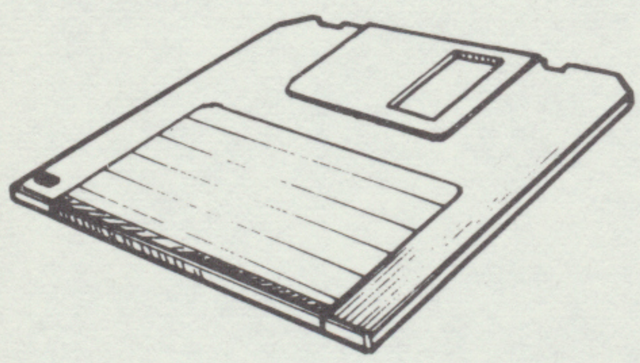


Höhere Programmiersprachen:

Eine höhere Programmiersprache ist eine Sprache, die vollkommen auf die eigentlichen Bearbeitungen ausgerichtet ist, die der Programmierer ausführen will.
Die Kennzeichnung 'höhere' wird vielleicht auf viele abschreckend wirken, weil man annehmen könnte, es handelt sich um etwas schwieriges wie beispielsweise 'höhere Mathematik'. Eher das Gegenteil ist der Fall: Programmierung mit einer solchen höheren Programmiersprache ist gerade relativ einfach.

Beispiele höherer Sprachen sind BASIC und PASCAL.
Es ist zu bedenken, daß jede Anweisung solch einer höheren Sprache vom Computer erst automatisch in Maschinenanweisungen umgesetzt wird, und dies sind die einzig brauchbaren Anweisungen für die zentrale Verarbeitungseinheit.

Operating System:



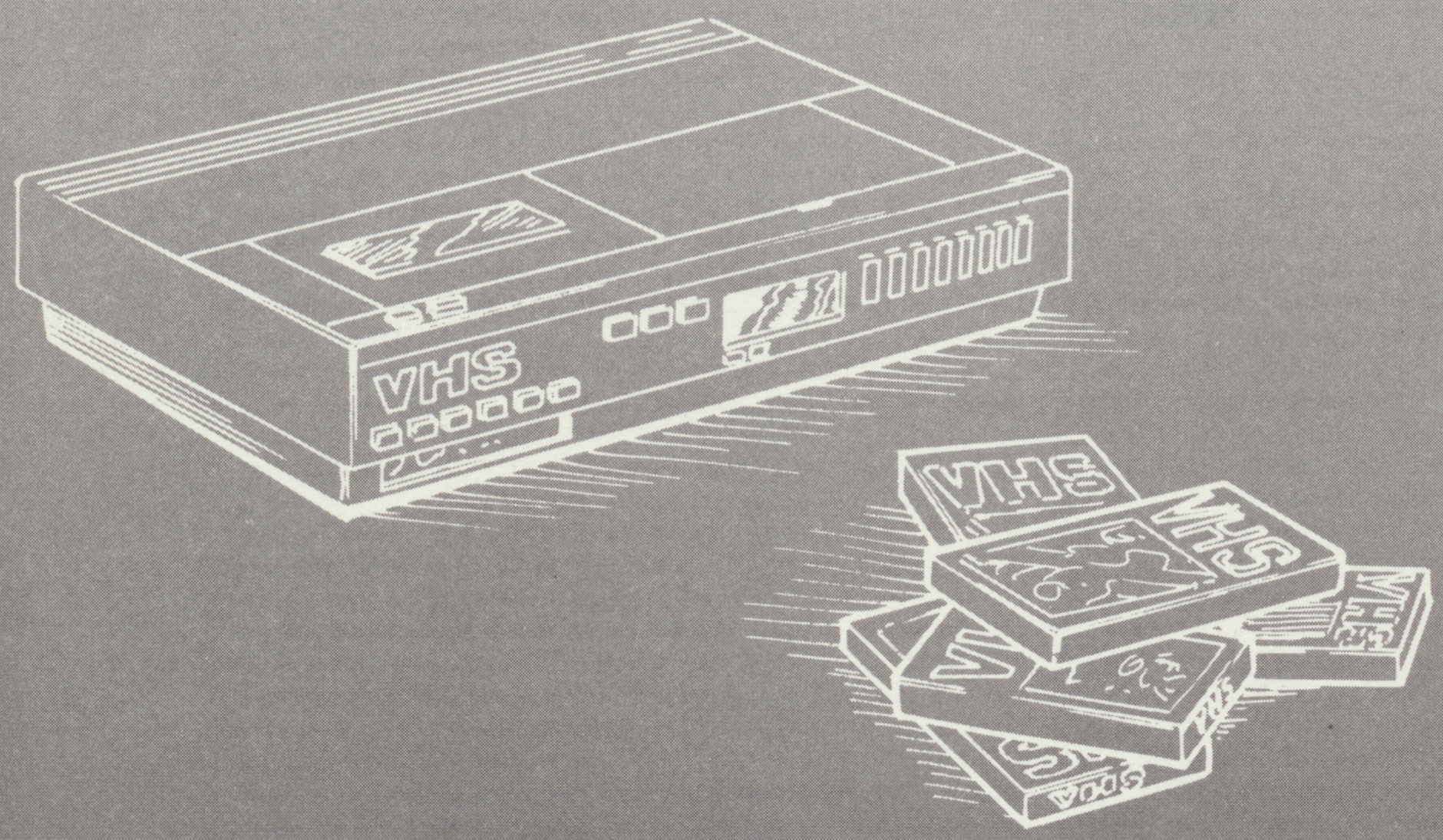
Ein Operating System ist eine Sammlung von Programmen, die der Computer zur Ausführung vieler Aufträge benötigt. Man denke etwa an

- das Einlesen von Programmen vom Floppy Disk oder von der Kassette;
- das Ausdrucken mit dem Drucker.

Diese Handlungen werden faktisch unter Steuerung von Teilprogrammen des Operating Systems erledigt. Das Operating System für MSX-Computer, die mit Disketten bzw. Floppy Disks arbeiten, trägt den Namen MSX-DOS. Der Ausdruck DOS ist übrigens die Abkürzung von Disc Operating System. Die einfachste Arbeitsweise mit Disketten geht aus von MSX Disk-BASIC (siehe Anhang J).

'Statt "Operating System" spricht man auch von Betriebssystem.'

Die Buchstaben MSX kennzeichnen einen Standard. In diesem Kapitel wird als erstes dargelegt, was wir unter dem Wort Standard zu verstehen haben. Hierbei ziehen wir Vergleiche mit den Standards von Videorecordern. Danach wird dargelegt, worauf sich der Standard MSX bezieht.
Schließlich erlauben wir uns Folgerungen über die möglichen Konsequenzen eines Standards.



2. MSX: ein Standard

Einleitung

Die Überschrift dieses Kapitels lautet: 'MSX: ein Standard'. Was wollen wir nun eigentlich genau mit 'Standard' zum Ausdruck bringen?
Die Bezeichnung Standard gibt an, daß verschiedene unabhängige Computerhersteller ihre Computer so bauen wollen, daß diese im Hinblick auf verschiedene Dinge völlig übereinstimmen.
Vergleichen Sie z.B. die Situation bei den Videorecordern. Auch da wurden Standards eingeführt. Das bekannteste (und am häufigsten verwendete) System ist VHS. Dies bedeutet, daß mit allen Videorecordern, auf denen VHS steht, Videobänder abgespielt werden können, auf denen ebenfalls VHS steht (unabhängig von der jeweiligen Marke).
Dasselbe gilt auch für MSX-Computer. Auf jedem Computer mit dem Kennzeichen

können wir Programme verarbeiten (die etwa auf Kassette oder Diskette angeboten werden), auf denen das gleiche Kennzeichen steht.
Standard heißt also in diesem Fall keineswegs, daß all die Computer völlig übereinstimmen. Auch bei Videorecordern gibt es schließlich Unterschiede. Die eine Marke bietet etwa die Möglichkeit, ein Bild stillstehen zu lassen und die andere nicht. So sehen wir beispielsweise auch, daß der eine MSX-Computerhersteller seinen Computer mit Extras ausstattet, die wir bei einem anderen MSX-Computer nicht finden. Beispiele solcher Extras sind:
- Angebot von Spezialprogrammen;
- besondere Anschlußmöglichkeiten, z.B. an einen Compact-Disc.

Nach dem vorigen Kapitel können wir etwas einfacher sagen, was nun ein MSX Standard genau ist.

Der Standard

Der Standard gibt an, daß der Computer auf jeden Fall mit folgenden Möglichkeiten ausgestattet ist:

- Einem Z80 Microprozessor. Wir haben zuvor bereits angegeben, daß dies der Bestandteil ist, mit dem die praktischen Arbeitsvorgänge ausgeführt werden.

- Einem Texas Instruments Video Chip (TMS 9929A oder einem gleichwertigen Typ). Es handelt sich hier wiederum um einen Chip, also eine zusammenhängende elektronische Schaltung, die als Einheit auf einer Siliziumscheibe hergestellt ist. Dieser Chip bestimmt, welche Möglichkeiten wir schließlich haben, um Abbildungen auf den Bildschirm zu bekommen. Diese Möglichkeiten sind also für jeden MSX-Computer gleich.

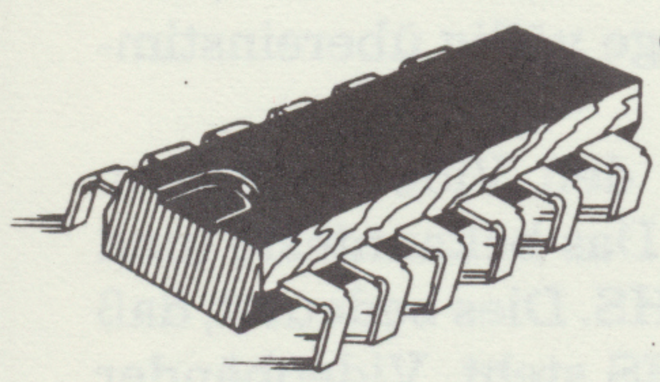


Abbildung eines Chips im Kunststoffgehäuse. Die tatsächlichen Maße der Schaltung sind nur wenige mm².

- Einem General Instruments Audiochip (AY-3-8912). Wieder ein Chip und in diesem Fall einer, der alle Geräuschmöglichkeiten unseres Computers bestimmt. MSX-Computer verdanken diesem Chip die Produktion sehr guter Töne über einen Bereich von 8 Oktaven, und zwar mit drei 'Stimmen' zugleich.

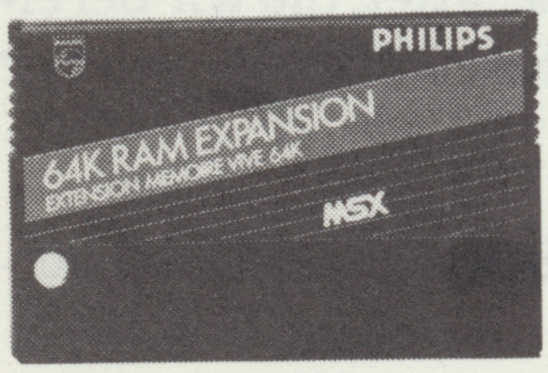
- 32 Kbyte ROM-Speicher. Hierin befindet sich u.a. das Programm, das unsere BASIC-Anweisungen umsetzt in eigentliche Anweisungen, die der Computer ausführt: die Maschinenanweisungen. Da dieses Programm zu einem Teil von amerikanischen Software-Entwickler Microsoft stammt, sprechen wir häufig von Microsoft-BASIC.

- Einem Bildschirm, auf dem Standard 40 Zeichen nebeneinander abgebildet werden können. Man spricht hier auch von einem '40- Spalten-Display'. Zur Wiedergabe von Figuren ist der Bildschirm in 256x192 Bildpunkte eingeteilt.

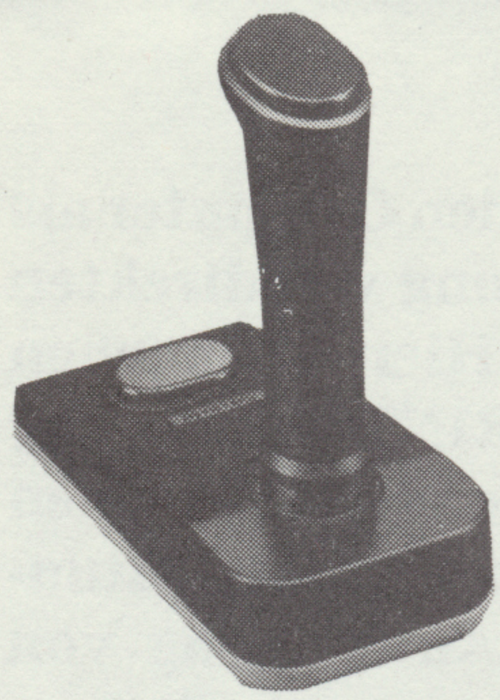
- Die Möglichkeit zur Verwendung von bis zu 16 verschiedenen Farben beim Aufbau solcher Figuren und dergleichen.

- Standardanschluß für einen Kassettenrecorder.

- Standardmöglichkeit für Floppy-Disc-Anschluß.



- Standardmöglichkeit zum Einschub sogenannter Cartridges in den MSX-Computer. Cartridges sind Kassetten, die ROM-Speicher enthalten, in denen fertige Programme untergebracht sind. Durch den Einsatz solch einer Cartridge verfügt man so-



fort über ein Programm. Es gibt sehr viele Cartridges mit Spielprogrammen, doch es gibt auch andere für ernsthaftere Zwecke, etwa Schulungsprogramme und dergleichen.

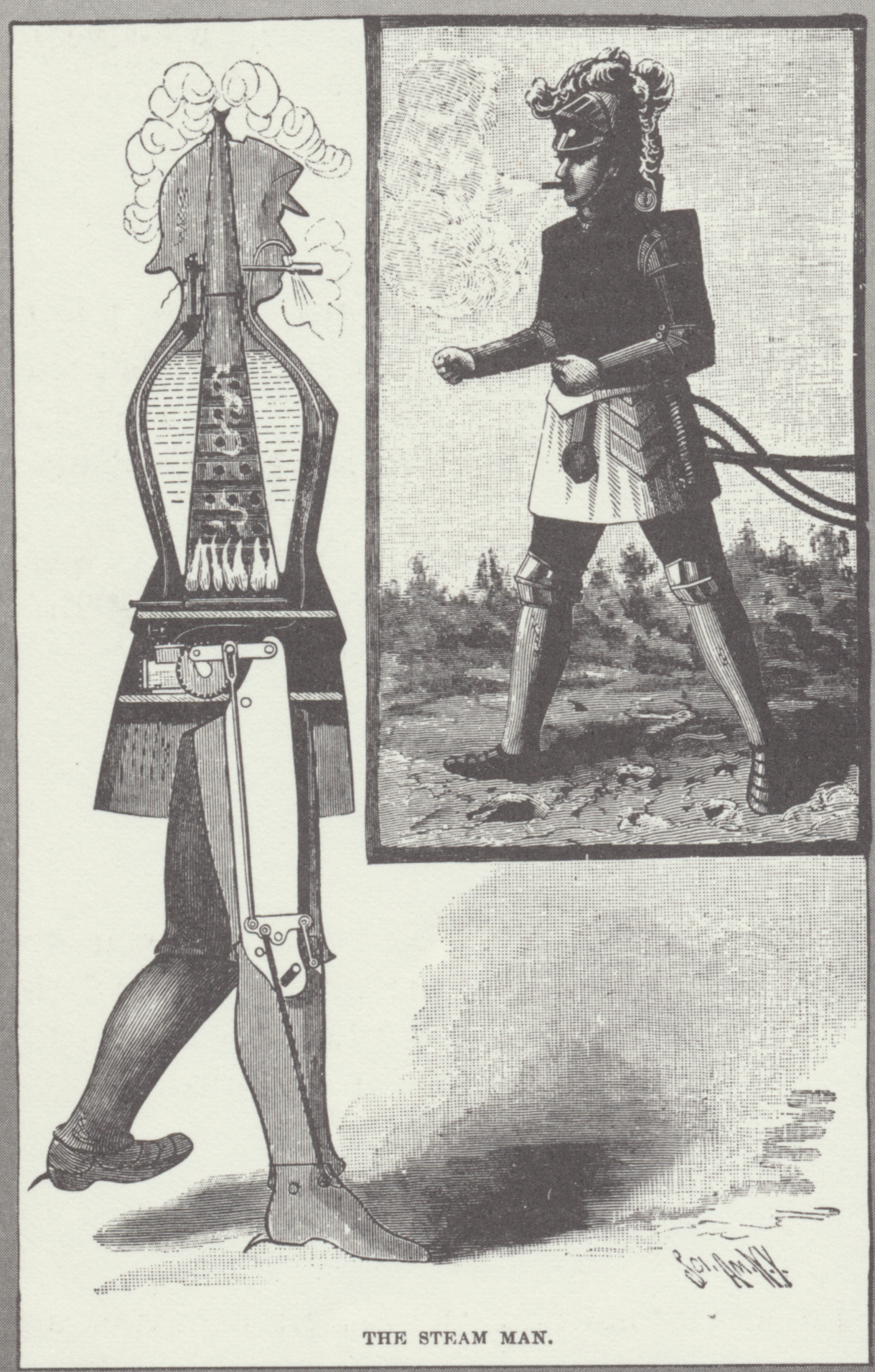
- Standardanschluß für mindestens einen Joy-Stick.

Die Folgen eines Standards

MSX-Computer werden von vielen unabhängigen Firmen angeboten. Neben Philips bringen auch japanische und koreanische Firmen MSX-Computer auf den Markt. Hieraus ergeben sich unseres Erachtens enorme Konsequenzen. Erstmals wird ein Standard (für Hardware wie Software) für Heimcomputer eingeführt. Es ist zu bedenken, daß sich Standard nicht nur auf BASIC bezieht. Viel wesentlicher ist, daß das Gesamtkonzept festgelegt ist. Software-Firmen werden enorme Mengen von Software entwickeln, darunter die bereits erwähnte Programmiersprache PASCAL.

In diesem Kapitel zeigen wir, wie wir den Computer als Rechner, d.h. als Gerät zur Durchführung von direkten Berechnungen gebrauchen können. Hierbei machen wir ständig von der sogenannten PRINT-Anweisung Gebrauch. Anschließend werden die verschiedenen Bearbeitungszeichen für mathematische Bearbeitungen gezeigt. Hierbei wird auch die Anwendung von Klammern behandelt. Schließlich zeigen wir in diesem Kapitel, daß wir auch Bildschirmtexte drucken lassen können.

In diesem Kapitel machen wir die ersten Schritte in die Computerpraxis.



THE STEAM MAN.

3. Die ersten Schritte: über PRINT und rechnerische Bearbeitungen

PRINT zur Durchführung von direkten Berechnungen

Mit dem Einschalten des Computers wird dieser sofort in den Zustand gebracht, in dem wir mit MSX-BASIC arbeiten können. Dies ist einer der großen Vorteile im Vergleich zu anderen höheren Programmiersprachen wie PASCAL. Wenn wir nämlich letztgenannte Programmiersprache gebrauchen wollen, müssen wir zuvor immer erst einige Handlungen vornehmen, auf die wir hier nicht weiter eingehen wollen.
Also gut, wir können sofort anfangen.
Um dies zu zeigen, tippen wir:

```
PRINT 2 + 3
```

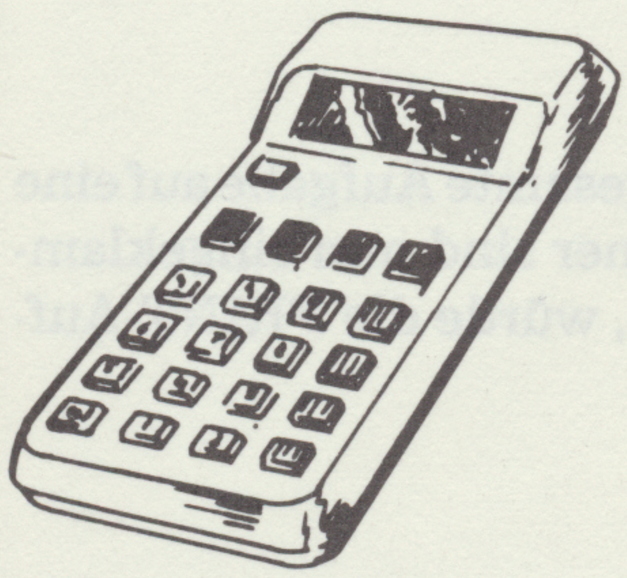
Auf dem Bildschirm erscheint nun der Ausdruck PRINT 2+3. Erst wenn wir die RETURN-Taste betätigen, reagiert der Computer auf diese Anweisung. Diese RETURN-Taste wird mit dem Wort RETURN wie mit dem Pfeilsymbol ← angegeben.

PRINT bedeutet 'drucken' oder in diesem Fall besser 'abbilden'. So sehen wir auch sofort nach dem Anschlagen der RETURN-Taste, daß etwas abgebildet wird, nämlich:

```
5
```

und der Ausdruck:

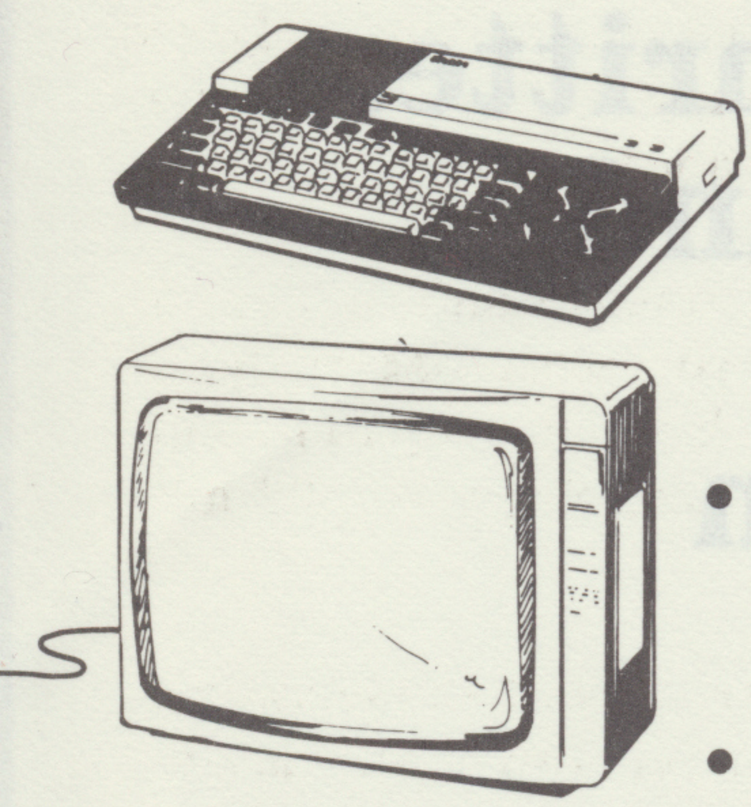
```
Ok
```



Es ist wohl klar, daß 5 die Antwort auf die verlangte Addition 2+3 ist. Der Ausdruck Ok gibt an, daß der Computer zur Ausführung einer neuen Aufgabe bereit ist. Der Leser begreift, daß wir den Computer auf diese Weise als eine Rechenmaschine gebrauchen.
Bei einem Taschenrechner tasten wir ein:

```
2+3=
```

und anschließend erscheint auf dem Display als Ergebnis: 5.



Beim Computer tasten wir ein:

```
PRINT 2+3
```

und nach Betätigung der RETURN-Taste erscheint das Ergebnis: 5.

Anmerkungen:

- Beim Computer dürfen wir keinesfalls tippen 2+3= (und etwa anschließend die RETURN-Taste anschlagen). Wenn wir das doch tun, erscheint die Mitteilung - natürlich in englisch - daß wir etwas falsch gemacht haben.
- Wenn wir beim Computer das Ergebnis sofort auf dem Bildschirm sehen wollen, dann muß die Programmzeile immer mit dem Ausdruck PRINT beginnen.

Noch einige Beispiele:

Multiplizieren

```
PRINT 5*36
```

Nach Betätigung der RETURN-Taste erscheint:

```
180
```

Aufgepaßt! Das 'Mal'-Zeichen ist hier ein Sternchen, damit keine Verwechslung mit dem Buchstaben x auftreten kann.

Dividieren

Der Auftrag:

```
PRINT 20/5
```

zeigt als Ergebnis:

```
4
```

Das Zeichen zum dividieren ist immer / und nicht etwa :

Klammern

Klammern dürfen wir gebrauchen, so wie wir dies auf der Schulbank gelernt haben. So wird etwa

$$\frac{5+15}{23\ (17+103)}$$

folgendermaßen eingegeben:

```
PRINT (5+15)/(23*(17+103))
```

Als erstes ist hier zu beachten, daß die gesamte Aufgabe auf eine Zeile geschrieben ist. Zähler und Nenner sind nun eingeklammert. Wenn wir das nicht getan hätten, würde die PRINT-Aufgabe folgendermaßen aussehen:

```
PRINT 5+15/23*(17+103)
```

Mit dieser Anweisung bekommen wir jedoch ein ganz anderes Ergebnis (und dies wird deutlich, wenn wir in der Folge die Prioritätsregeln für Berechnungen behandeln).

Wir sehen außerdem, daß wir beim Nenner zwischen 23 und 'Klammer auf' ein Multiplikationszeichen gesetzt haben, denn es handelt sich bei der ursprünglichen Bezeichnung des Nenners '23(17+103)' schließlich um ein Malnehmen von 23 mit (17+103).

Reihenfolge der Bearbeitungen

Wir haben in der Schule die Regel gelernt, daß beispielsweise das Malnehmen vor dem Teilen stattfinden muß oder, wissenschaftlicher gesagt, die Multiplikation Priorität hat vor der Division. Bei MSX-BASIC sind die folgenden Prioritätsregeln zu beachten:

1. Als erstes werden die eingeklammerten Aufgaben verarbeitet;

2. Multiplizieren und Dividieren haben gleiche Priorität und gehene vor Addieren und subtrahieren, die ebenfalls gleiche Priorität haben;

3. Bearbeitungen mit gleicher Priorität werden von links nach rechts erledigt.
   Beispiel:

```
PRINT            15/3*5
wird zu          5*5
und damit zu     25
```

Das Ergebnis ist also 25 (und nicht etwa 1!).
Im Anhang F ist eine Übersicht aller Prioritätsregeln wiedergegeben.

Text

Die Textverarbeitung ist eines der wichtigsten Einsatzgebiete von Computern. Wir werden in der Folge noch sehen, welche Möglichkeiten BASIC hier bietet. Vorläufig zeigen wir nur, daß wir auch einen bestimmten Text auf dem Bildschirm wiedergeben können.
Den abzubildenden Text tippen wir immer zwischen Anführungszeichen. Beispiel:

```
PRINT "DIES IST EIN BEISPIEL"
```

ergibt:

```
DIES IST EIN BEISPIEL
```

Bei dem Ergebnis sind also keine Anführungszeichen zu sehen.

In diesem Kapitel lernen wir ein BASIC-Programm kennen. Ein BASIC-Programm ist eine Reihe von Anweisungen. Um deutlich zu machen, daß es sich um ein Programm handelt, gebrauchen wir sogenannte Zeilennummern. Diese Nummern bestimmen die Reihenfolge, in der die Anweisungen zu erledigen sind.
Danach schließen wir erste Bekanntschaft mit dem RUN-Befehl. Mit diesem Befehl erhält der Computer den Auftrag, das im Speicher enthaltene Programm auch wirklich auszuführen. Schließlich geben wir in einer Nachbetrachtung noch eine Kurzdarlegung über verschiedene Programmiersprachen. Kurz und gut, in diesem Kapitel gehen wir tiefer auf die Software-Möglichkeiten unseres Computers ein.

2 Software und Standard-Software

Wie in vielen Bereichen eines raschen technologischen Fortschritts bleibt die Begriffsbildung auch in der automatischen Datenverarbeitung hinter der Entwicklung der technischen Anwendungen zurück und ist nicht einheitlich. Das gilt insbesondere für den Bereich der Software, der in gewissen Zeitabständen – verursacht durch technologische Veränderungen – Redefinitionen unterliegt. Zum Verständnis und zur Gliederung des Untersuchungsgegenstandes ist daher eine vorausgehende begriffliche Abgrenzung notwendig.

2.1 Softwarebegriffe und Softwarearten

In der Literatur wie auch in der Praxis werden in der Regel verschiedene Definitionen zur Abgrenzung des Begriffes Software angetroffen, die z. T. historisch bedingt sind und dann beibehalten wurden:

a) Einerseits werden darunter alle dem Anwender zur Verfügung gestellten Mittel und Methoden zur Nutzung eines ADV-Systems verstanden. Dazu zählen alle Vorhaben, die den Betrieb der Hardware ermöglichen und nicht selbst Hardware sind, wie Programme, Programmiersprachen, Problemlösungs- und Organisationsleistungen, Dokumentation, Schulung, Beratung u. ä.

b) Andererseits können als Software alle einem ADV-Anwender zur Verfügung gestellten Programme und Programmiersprachen sowie seine eigenentwickelten Anwendungsprogramme gelten.

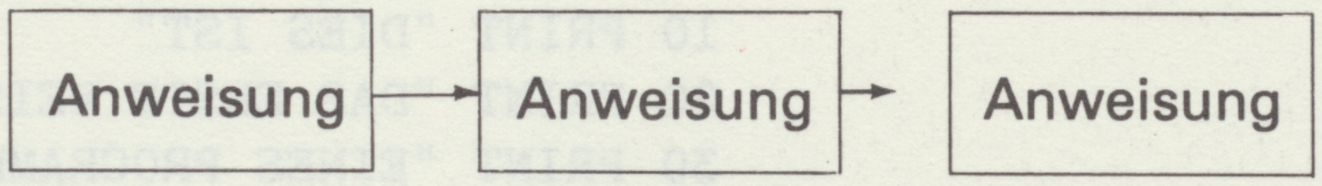
c) Ähnlich wird als Software die Gesamtheit aller Programme, die für ein ADV-System zur Verfügung steht, bezeichnet (wobei allerdings die mikroprogrammierten Hardware-Elemente ausgenommen sind).

Der Begriffes Software wird für die weiteren

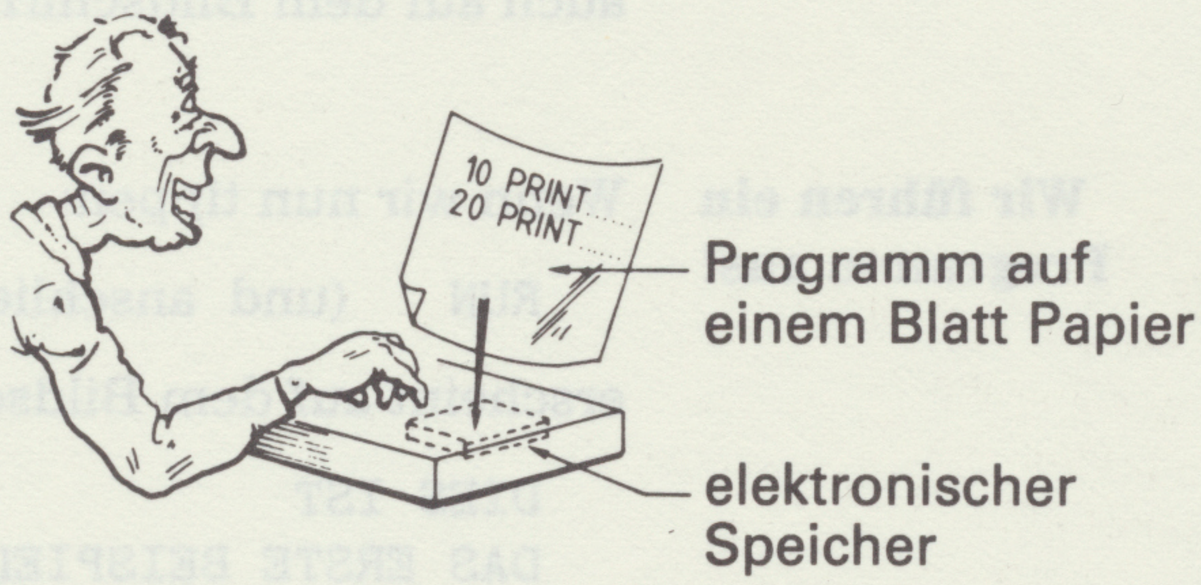
4. Und jetzt ein BASIC-Programm

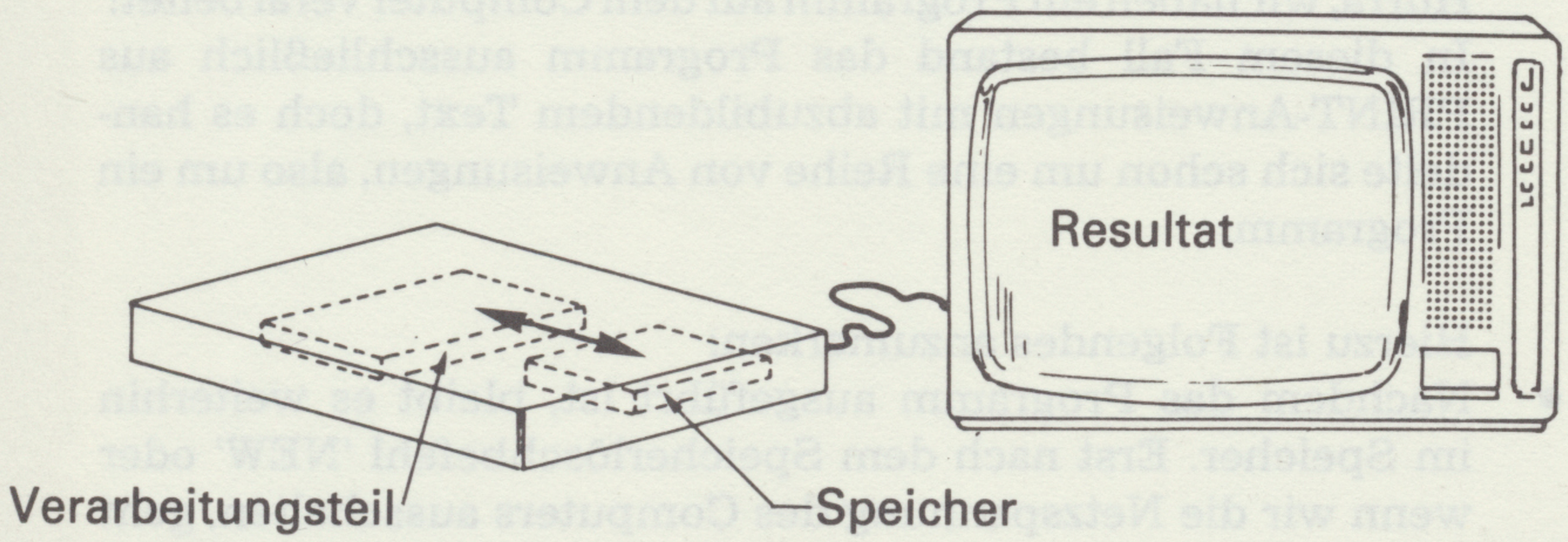
Ein BASIC-Programm: Anwendung von Zeilennummern

In Kapitel 1 wurde erklärt, daß ein Programm nichts anderes ist, als eine Serie von Anweisungen, die nach einem bestimmten Start-Auftrag automatisch vom Computer ausgeführt wird. Schematisch:



Im einfachsten Fall können wir ein Programm, sofern es sich um einen Text handelt, einfach auf der Tastatur tippen. Dies bedeutet, daß solch ein Programm bzw. eine Reihe von Anweisungen zuerst eingespeichert wird.
Bildlich:





Wie können wir nun dafür sorgen, daß die Anweisungen eines Programmes zuerst in den Speicher des Computers eingegeben und nicht sofort ausgeführt werden? Die Antwort ist einfach: Durch Angabe von Nummern vor den Anweisungen.
Wir werden dies in der Folge durch ein Beispiel illustrieren.

Ein Beispiel: Wir gehen einmal davon aus, daß der Computer als Programm nacheinander die folgenden drei Anweisungen ausführen soll:

```
PRINT "DIES IST"
PRINT "DAS ERSTE BEISPIEL"
PRINT "EINES PROGRAMMES"
```

Wir müssen nun beim Eingeben dieses Programmes - es handelt sich schließlich um eine Reihe von Anweisungen - vor jede Anweisung eine Nummer setzen.
Wir numerieren im allgemeinen mit 10, 20, 30 usw.
Also tippen wir:

```
10 PRINT "DIES IST"
20 PRINT "DAS ERSTE BEISPIEL"
30 PRINT "EINES PROGRAMMES"
```

Am Ende jeder Zeile geben wir ein RETURN als Zeichen, daß die Zeile vollständig eingegeben ist (nicht vergessen: auch nach der letzten Zeile RETURN betätigen!).
Nachdem auf diese Weise alle Zeilen eingegeben sind, ist das Programm insgesamt eingespeichert. Das sehen wir übrigens auch auf dem Bildschirm.

Wir führen ein Programm aus! Wenn wir nun tippen:

```
RUN    (und anschließend RETURN)
```

erscheint auf dem Bildschirm:

```
DIES IST
DAS ERSTE BEISPIEL
EINES PROGRAMMES
```

Hurra, wir haben ein Programm auf dem Computer verarbeitet!
In diesem Fall bestand das Programm ausschließlich aus PRINT-Anweisungen mit abzubildendem Text, doch es handelte sich schon um eine Reihe von Anweisungen, also um ein Programm.

Hierzu ist Folgendes anzumerken:

- Nachdem das Programm ausgeführt ist, bleibt es weiterhin im Speicher. Erst nach dem Speicherlöschbefehl 'NEW' oder wenn wir die Netzspannung des Computers ausschalten, geht das Programm verloren.

- Die Nummern geben zugleich die Reihenfolge an, in der die Anweisungen ausgeführt werden, und zwar von der niedrigsten bis zur höchsten Zahl. Wir hätten auch eingeben können:

```
10 PRINT "DIES IST"
30 PRINT "EINES PROGRAMMES"
20 PRINT "DAS ERSTE BEISPIEL"
```

  Das Endergebnis würde das gleiche sein, weil die Zeilennummern das so vorschreiben. Sofort nach dem Eintippen würde der Computer sie nämlich in die richtige Reihenfolge bringen.
- Statt von Nummern, sprechen wir im allgemeinen von Zeilennummern. Als Zeilennummern dürfen wir nur ganze Zahlen verwenden (von 0 bis 65529).
- Daß wir jetzt mit 10, 20, 30 usw. numerieren, hat den Vorteil, daß wir später noch Anweisungen einfügen können. In Kapitel 5 behandeln wir genauer, wie das vor sich geht.

Nachbetrachtung

Das war unsere erste Bekanntschaft mit einem Programm. Wir können aber wohl kaum behaupten, daß wir das eigentliche Programmieren kennengelernt haben. Wir wissen also noch nicht, wie wir mit einem Programm ein bestimmtes Resultat erzielen können. Zuerst müssen wir noch weitere Anweisungen durchnehmen, um einen tieferen Einblick in das Programmieren zu gewinnen.
Wir wollen jedoch noch einen Augenblick bei dem einen gebrachten Beispiel verweilen, weil dies ein typisches Beispiel einer Anweisung in einer höheren Programmiersprache ist. Betrachten wir nochmals die PRINT-Anweisung:

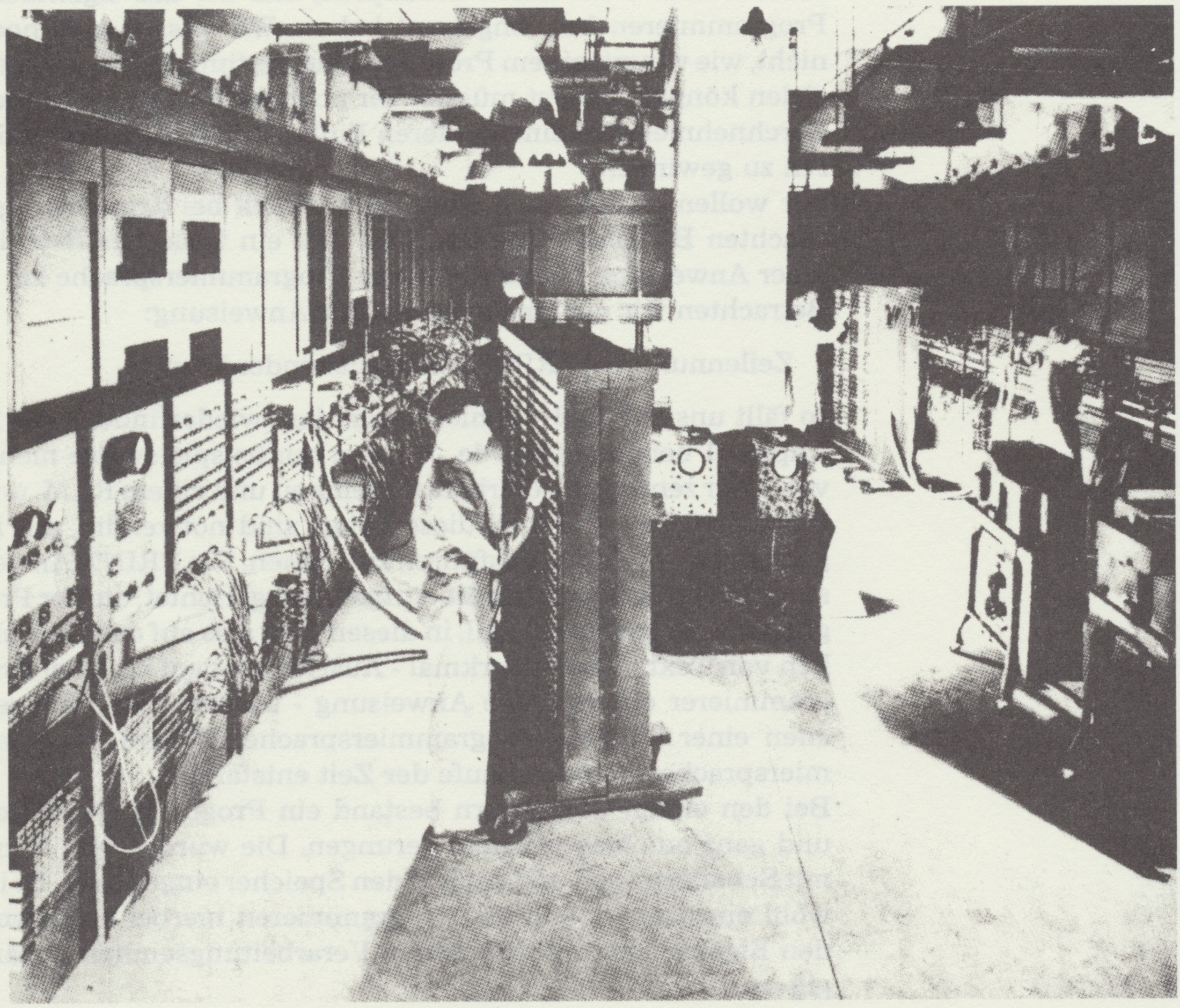
Zeilennummer PRINT ‘abzudruckender Text’

so fällt uns auf, daß wir nichts von dem wiederfinden, was in Kapitel 1 behandelt wurde. Wir merken beispielsweise nichts von einer zentralen Verarbeitungseinheit und einem RAM- und ROM-Speicher. Doch all diese Dinge sind notwendig, um irgendeine Anweisung ausführen zu können. Die PRINT-Anweisung ist voll und ganz auf die Aufgabe ausgerichtet, die der Programmierer ausführen will, in diesem Fall also auf das Abdrukken von Text. Dieses Merkmal - Ausrichtung auf die vom Programmierer eingegebene Anweisung - ist eben das Kennzeichen einer höheren Programmiersprache. Höhere Programmiersprachen sind im Laufe der Zeit entstanden.
Bei den ersten Computern bestand ein Programm noch voll und ganz aus Maschinenkodierungen. Die wurden von Hand mit Schaltern nach und nach in den Speicher eingegeben. Es ist wohl einzusehen, daß das Programmieren hierbei völlig von den Eigenschaften der zentralen Verarbeitungseinheit abhängig war.

Vom ersten Augenblick an suchte man nach einfacheren Programmiermethoden. So entstanden als erstes die sogenannten Assembler-Sprachen. Dies sind Sprachen, bei denen die Maschinenanweisungen in kurzen, leicht zu behaltenden Buchstabenkodes eingegeben werden. Danach entstanden die ersten höheren Programmiersprachen wie FORTRAN und BASIC. Beide Sprachen sind noch quicklebendig. FORTRAN wird vor allem von Rechenzentren angewandt, in denen umfassende mathematische Berechnungen durchgeführt werden. BASIC hat sich vor allem in Heimcomputern einen sehr festen Platz erobert.

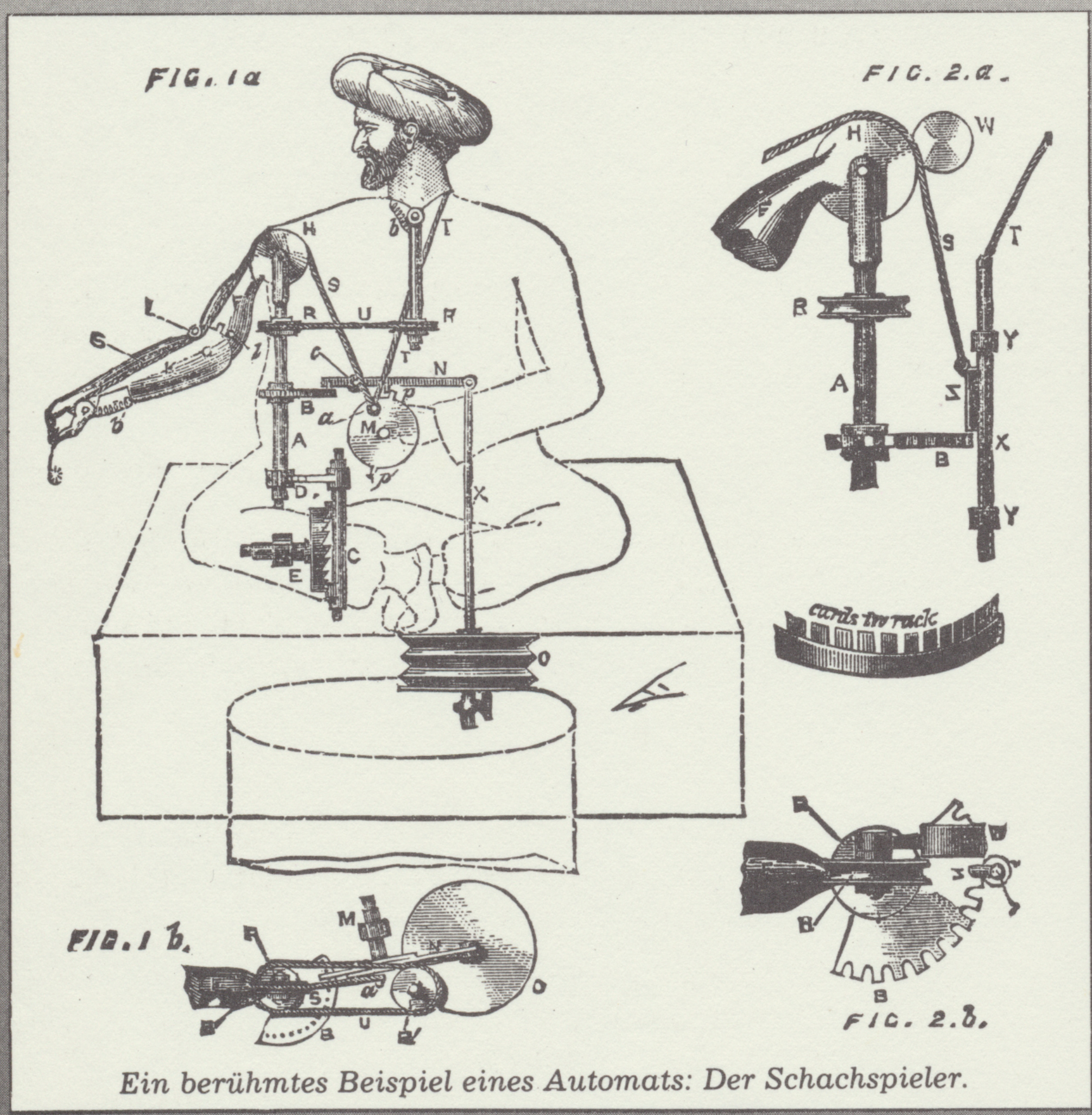
Die Entwicklungen kennen natürlich keinen Stillstand. Neue Programmiersprachen werden vorgestellt. Man sieht ferner die Entwicklung neuer 'Sprachen' für ganz besondere Einsatzbereiche. Ein sehr bekannter Bereich ist beispielsweise die Verwaltung sehr großer Mengen von Daten (Dateien).

Der ENIAC von 1946



Diese 'data-base-Sprachen' stehen übrigens auch für MSX-Computer zur Verfügung. Schließlich wollen wir noch anmerken, daß die Bedeutung von Maschinenkodes und Assembler-Sprachen auch nach Einführung der höheren Programmiersprachen keineswegs verschwunden ist. Letzten Endes arbeitet jeder Computer mit Maschinenkodes. Bei einer ersten Bekanntschaft ist dies jedoch eine kaum verdauliche Materie, und darauf werden wir hier nicht tiefer eingehen.

In diesem Kapitel werden vor allem verschiedene Befehle beschrieben: RUN Programmlauf, LIST zur Aufzeichnung der Programmanweisungen auf dem Bildschirm, NEW zum Löschen eines Programmes. Wir besprechen auch den Befehl AUTO zur automatischen Zeilennumerierung, RENUM zur automatischen Umnumerierung von Zeilennummern und DELETE zum automatischen Löschen bestimmter Programmteile. Schließlich sehen wir noch, wie wir Zeilen hinzufügen, einfügen und abändern können. Dieses Kapitel verdeutlicht nochmals, daß der Computer wirklich ein Automat ist, der ohne menschlichen Einsatz viele Handlungen ausführen kann.



Ein berühmtes Beispiel eines Automats: Der Schachspieler.

5. RUN, LIST, NEW, AUTO, RENUM und DELETE

Einleitung Bisher haben wir gelernt, wie wir ein Programm eingeben können. Nach dem Eintippen des Befehls RUN beginnt das Programm auch wirklich zu laufen. An dieser Stelle ist es wohl angebracht, auch auf den Unterschied zwischen einer Anweisung und einem Befehl einzugehen.
Eine Anweisung ist immer Bestandteil eines Programmes, während ein Befehl dem Computer sagt, daß er eine bestimmte Handlung unverzüglich auszuführen hat.
Schematisch:

Anweisung: ↓	Befehl: ↓
Eine Reihe von Anweisungen bildet ein Programm; während des Tippens werden diese Anweisungen gespeichert.	Der Computer muß sofort nach dem Eingeben des Befehls eine bestimmte Aufgabe ausführen.

Die bekanntesten Befehle sind RUN, LIST und NEW. Wir werden sie in der Folge besprechen. Weiter besprechen wir noch eine Anzahl von Befehlen, die als Zusatz erachtet werden können: nicht unbedingt notwendig, aber mit Annehmlichkeiten verbunden.

RUN Mit dem RUN- Befehl sagen wir dem Computer, daß ein eingespeichertes Programm sofort ausgeführt werden muß. Nehmen wir einmal an, wir haben zuvor eingetippt:

```
10 PRINT "DIES IST EIN"
20 PRINT "PROGRAMM DAS AUS"
30 PRINT "3 ZEILEN BESTEHT"
```

Nach dem Tippen von RUN (natürlich mit anschließendem RETURN) erscheint:

```
DIES IST EIN
PROGRAMM DAS AUS
3 ZEILEN BESTEHT
```

Nach RUN kann man auch eine Zeilennummer angeben. Dann wird das Programm von dieser Zeile ausgeführt.
Beispiel:

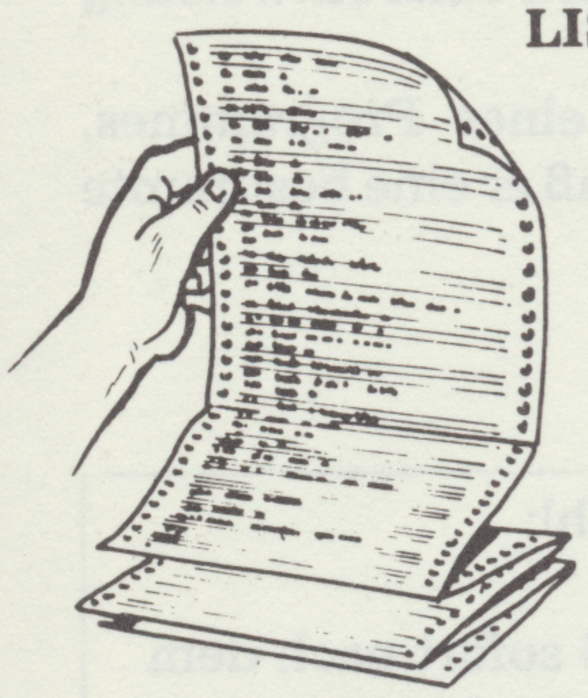
```
RUN 20     (anschließend RETURN)
```

erbringt als Ergebnis:

```
PROGRAMM DAS AUS
3 ZEILEN BESTEHT
```

LIST Programmieren ist eine vor allem dynamische Tätigkeit. Ein Programm ist selten sofort in Ordnung. Wir müssen Anweisungen verändern, einfügen, weglassen, und häufig ist es nötig, daß wir das ursprüngliche Programm wieder auf dem Bildschirm erscheinen lassen.
Zu diesem Zweck bietet der Computer den LIST-Befehl. Gehen wir davon aus, daß das eben gezeigte Programm noch im Speicher ist.
Wenn wir jetzt eingeben:

```
LIST     (anschließend RETURN)
```



Einen Programm-abdruck auf Papier nennt man 'listing'; das Wort kommt von LIST

dann erscheint das vollständige Programm auf dem Bildschirm, so daß wir sehen:

```
10 PRINT "DIES IST EIN"
20 PRINT "PROGRAMM DAS AUS"
30 PRINT "3 ZEILEN BESTEHT"
```

Durch eine Zeilennummer nach LIST können wir Teile des Programmes aufrufen:

```
LIST 20     : nur Zeile 20
LIST 20-30  : Zeile 20 bis 30
LIST -20    : alles bis einschließlich Zeile 20
LIST 20-    : alles ab Zeile 20
```

NEW Der Befehl NEW ist der absolut gefährlichste BASIC-Befehl. Mit dem Befehl NEW geben wir nämlich an, daß das eingespeicherte Programm gelöscht werden soll. Was aber einmal gelöscht wird, ist endgültig weg!

Diesen Befehl also nur dann anwenden, wenn es unbedingt nötig ist, also:

- wenn das zu löschende Programm absolut uninteressant ist;
- wenn Sie das Programm schon auf der Kassette (oder Diskette) abgespeichert haben;
- erst nachdem Sie das Programm durch einen entsprechenden Befehl auf der Kassette (oder Diskette) abgespeichert haben (siehe Anhang D).

AUTO

MSX-BASIC hat viele Möglichkeiten, die einerseits nicht unbedingt nötig sind, aber andrerseits manche Annehmlichkeiten bieten. Eine dieser Möglichkeiten ist der AUTO-Befehl. Nach AUTO kann man eine oder zwei Zahlen angeben.
Beispiel:

`AUTO 10` : In diesem Fall produziert der Computer automatisch die Zeilennummern 10, 20, 30 usw.

`AUTO 100, 10` : Hier werden zwei Zahlen angeboten. Die erste Zahl gibt die erste Zeilennummer an und die zweite Zahl, in welchen Schritten die nächsten Zeilennummern generiert werden sollen. Bei diesem Beispiel produziert also der Computer die Zeilenreihe 100, 110, 120, 130 usw.

Wenn der Computer die automatische Numerierung einstellen soll, betätigen wir bei eingedrückter CTRL-Taste die Taste STOP. Diese Handlung geben wir immer auf folgende Weise an:

CTRL/STOP

Geben Sie als erste Übung einmal das oben gegebene Programm mit Hilfe des AUTO-Befehls ein.

Zusatz, Einfügen, Umnumerierung von Zeilen und RENUM-Befehl

Damit wir den Wert des RENUM-Kommandos erkennen, zeigen wir wieder erst ein Programm:

```
10 PRINT "DIES IST EIN"
20 PRINT "PROGRAMM"
30 PRINT "ZUR ILLUSTRATION"
```

Nachdem wir dieses Programm einmal eingegeben haben, können wir einfach neue Anweisungen hinzufügen oder einfügen. Das Hinzufügen geschieht mehr oder weniger selbstverständlich: Sie wählen eine Zeilennummer, die auf die letzte Zeilennummer folgt.

Wenn wir beispielsweise eingeben:

```
40 PRINT "DES ARBEITENS"
50 PRINT "MIT ZEILENNUMMERN"
```

dann werden diese beiden Zeilen direkt an die voraufgegangenen angehängt. Zur Einfügung einer Zeile wählen wir eine Zeilennummer, die zwischen den Zeilennummern liegt, zwischen denen die neue Zeile eingefügt werden muß.
Wenn wir beispielsweise eingeben:

```
15 PRINT "KURZES EINFACHES"
```

und anschließend den LIST-Befehl geben, so sehen wir auf dem Bildschirm:

```
10 PRINT "DIES IST"
15 PRINT "EIN KURZES EINFACHES"
20 PRINT "PROGRAMM"
30 PRINT "ZUR ILLUSTRATION"
40 PRINT "DES ARBEITENS"
50 PRINT "MIT ZEILENNUMMERN"
```

Zeile 15 ist also an der richtigen Stelle eingefügt.
Dabei sehen wir zugleich, wie nützlich es war, zu Beginn grundsätzlich in der Reihenfolge 10, 20, 30 usw. zu numerieren. Bei dieser Art der Numerierung ist es schließlich einfach, neue Zeilen einzufügen.
Das Einfügen neuer Zeilen ergibt sich in der Praxis so oft, daß häufig eine Vielfalt an Zeilennummern zu sehen ist.

Um nach dem Einfügen wieder eine geregelte Numerierung zu bekommen, können wir den RENUM-Befehl anwenden. Hierbei ergeben sich u.a. folgende Möglichkeiten:

`RENUM`: Wird nach RENUM keine Zahl angegeben, so numeriert der Computer alle Zeilen wieder im Zehnerrhythmus mit 10, 20, 30 usw.

`RENUM 100`: Wie bei RENUM, doch jetzt wird die Reihe 100, 110, 120, 130 usw. Bei RENUM 200 wird die Reihe 200, 210, 220, 230 usw.

`RENUM 100, 40, 10`: Jetzt werden ab Zeile 40 alle Zeilen erneut numeriert in der Reihenfolge 100, 110, 120 usw.

Wir probieren den RENUM-Befehl aus und tippen dazu:

```
RENUM
```

und sehen über den LIST-Befehl nunmehr:

```
10 PRINT "DIES IST"
20 PRINT "EIN KURZES EINFACHES"
30 PRINT "PROGRAMM"
40 PRINT "ZUR ILLUSTRATION"
50 PRINT "DES ARBEITENS"
60 PRINT "MIT ZEILENNUMMERN"
```

Der Computer hat also die Zeilen gemäß der Reihe 10, 20, 30 usw. einwandfrei neu numeriert.

DELETE

Als letzten behandeln wir in diesem Kapitel den DELETE- Befehl. Damit können wir eine oder mehrere Zeilen aus einem Programm entfernen. Der Aufbau ist fast so wie beim LIST-Befehl. Beispiele:

```
DELETE 20      : beseitigt Zeile 20
DELETE 20–40   : beseitigt Zeilen 20-40
DELETE –40     : beseitigt alle Zeilen bis 40
DELETE 20–     : im Gegensatz zu LIST ist dies
                 bei DELETE nicht erlaubt.
```

Auch dieses Programm erproben wir an unserem Programmbeispiel.
Wir geben ein:

```
DELETE 40–60
```

und nach dem LIST-Befehl zeigt der Computer:

```
10 PRINT "DIES IST"
20 PRINT "EIN KURZES EINFACHES"
30 PRINT "PROGRAMM"
```

Fehlerbehebung

Wir haben oben bereits einige Möglichkeiten einer Programmänderung besprochen. In diesem Abschnitt zeigen wir, was wir machen können, wenn wir einen Fehler gemacht haben.
Die offenbar einfachste Lösung wäre es, die fehlerhafte Zeile erneut einzugeben, jedoch diesmal fehlerlos.

Beispiel:
Eingeben:

```
20 PRINT "EIN KURXES EINFACHES"
```

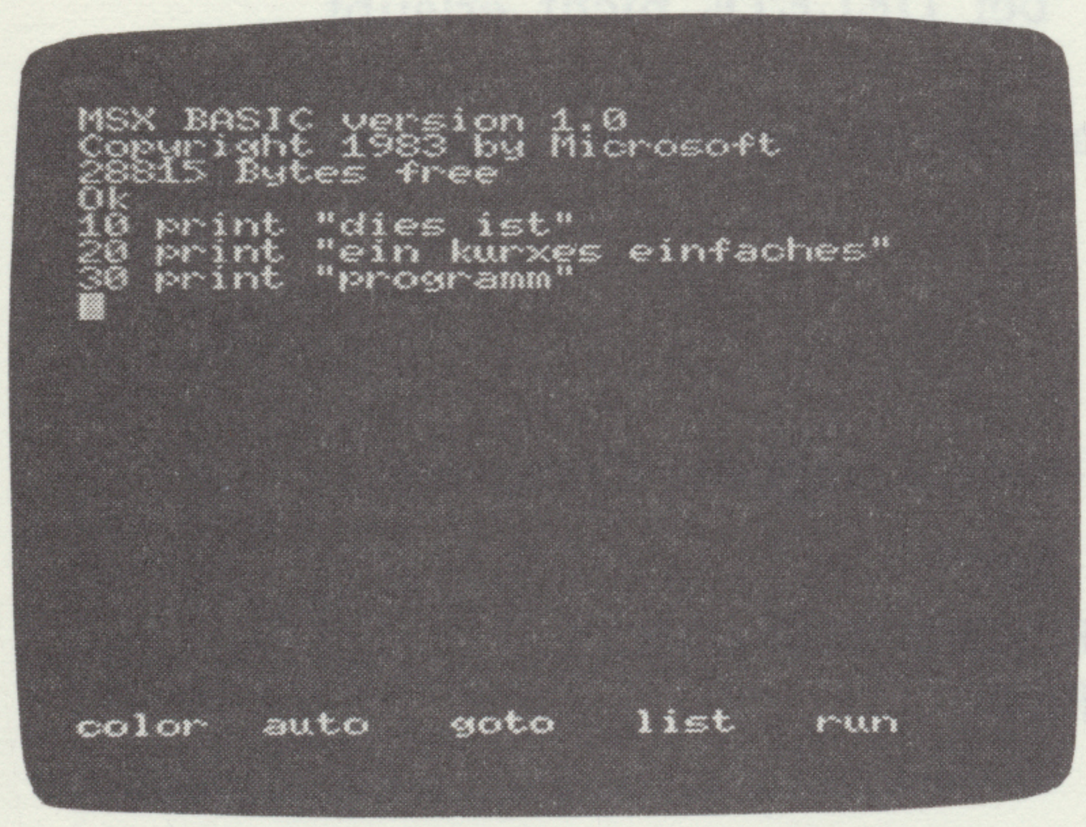
Nach dem LIST-Befehl sehen wir, daß diese Zeile mit dem auffälligen Fehler ins Programm aufgenommen wurde. Wir können diesen Fehler also berichtigen, indem wir diese Zeile fehlerfrei eingeben.
Eine elegantere Lösung bietet die Benutzung der Cursor-Tasten - dies sind die Tasten mit den Pfeilen an der rechten Seite der Tastatur. Wenn sie auf eine dieser Tasten drücken, können Sie sehen, wie sich das weiße Cursor-Symbol über den Bildschirm bewegt.
Wir nennen dieses Blöckchen Cursor und daher auch die Tasten mit den Pfeilen Cursorsteuertasten.

Versetzen Sie den Cursor so, daß er sich genau auf dem Fehler befindet:

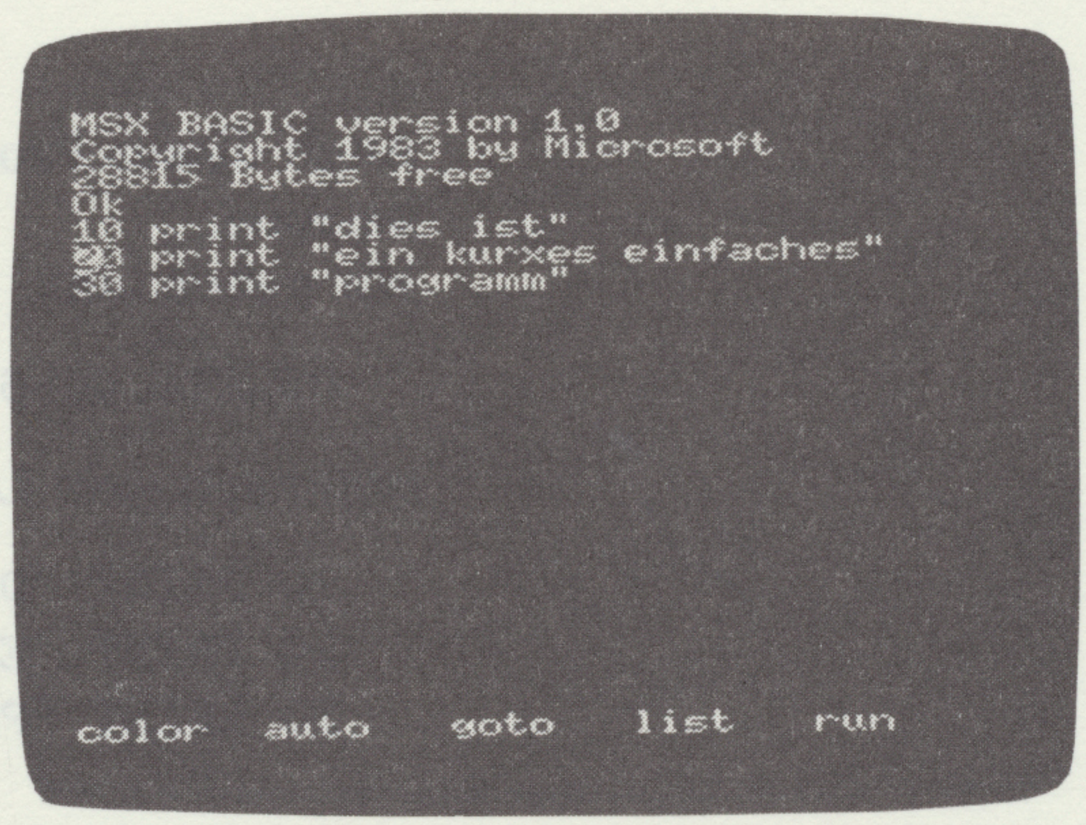
```
20 PRINT "EIN KUR[X]ES EINFACHES"
                 ↑
```

Dort steht der Cursor.

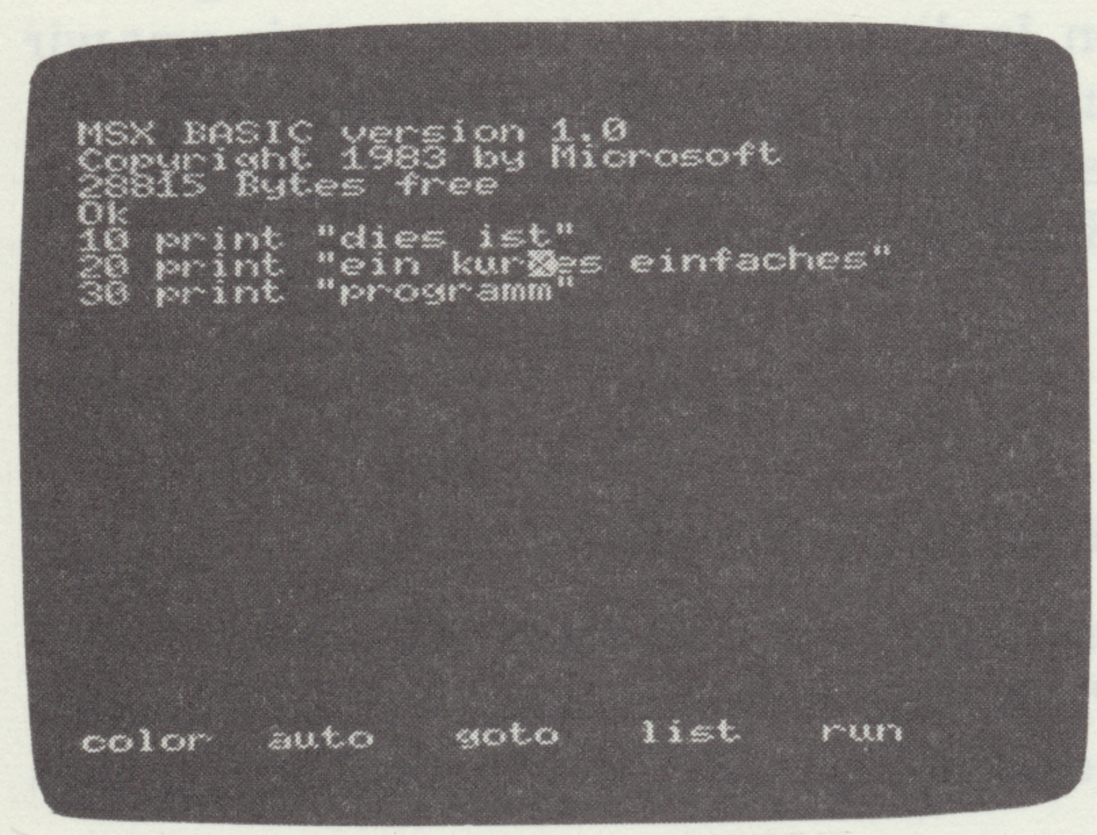
Nun tippen wir das 'Z' und betätigen dann die RETURN-Taste, wodurch der Cursor wieder unter das Programm geht. Siehe da: der Fehler ist beseitigt!
Wir können eine Cursor-Taste auch gedrückt halten, so daß der Effekt wiederholt wird. Dieser 'Wiederholungseffekt' betrifft übrigens auch alle übrigen Tasten.



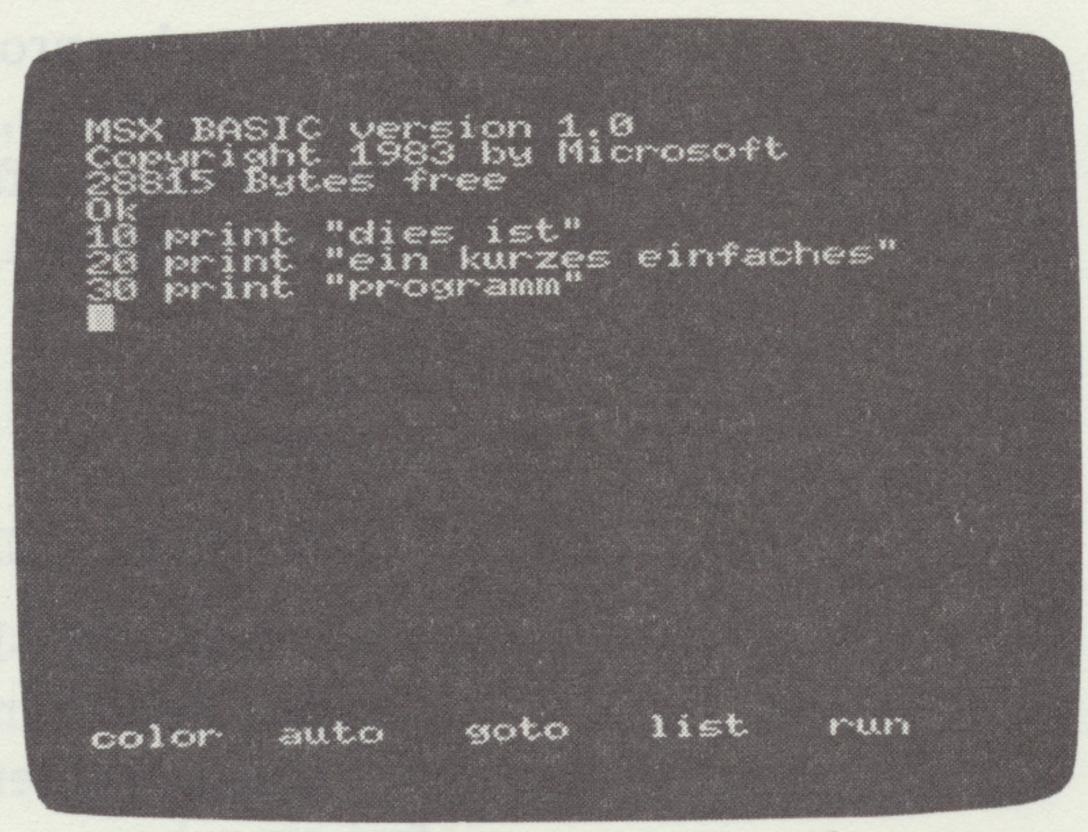
1 Zustand vor Fehlerberichtigung *



2 Cursor zur fehlerhaften Zeile versetzt



3 Cursor zum fehlerhaften Buchstaben versetzt: richtigen Buchstaben tippen und RETURN-Taste betätigen



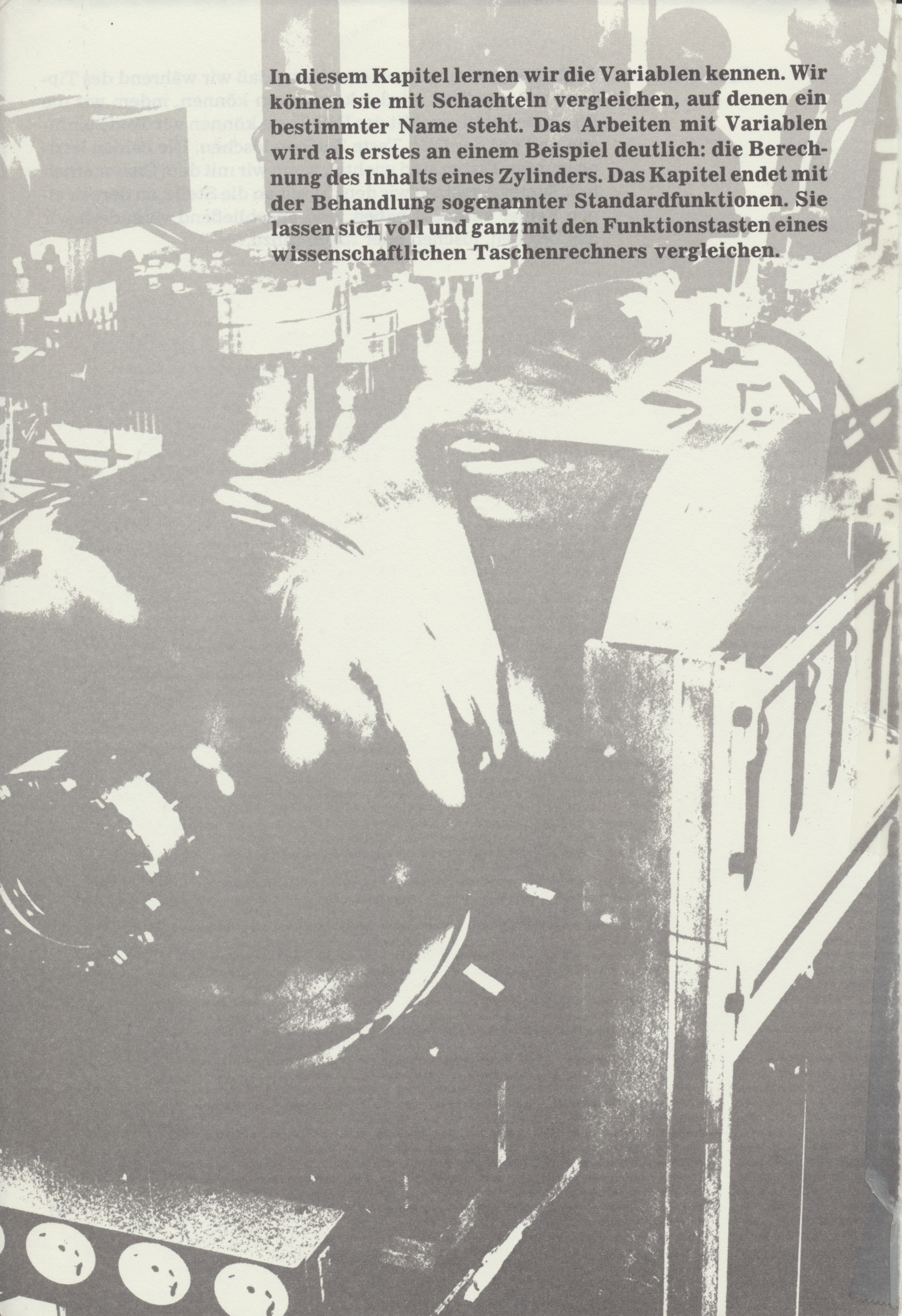
4 Cursor wieder ans Ende versetzt; der Fehler ist berichtigt

* Je nach verwendetem MSX-Computer kann die Anzahl freier Speicherstellen (auf dem Schirm 'Bytes free') unterschiedlich sein.

Spezialtasten

Schließlich wäre noch zu erwähnen, daß wir während des Tippens unseren Text wieder korrigieren können, indem wir die BS-Taste betätigen. Mit der INS-Taste können wir Texte einfügen und mit der DEL-Taste Zeichen löschen. Die beiden letztgenannten Tasten sind wichtig, wenn wir mit dem Cursor arbeiten. Zuerst versetzen wir den Cursor an die Stelle, an der wir einen Fehler entdeckt haben, und anschließend verändern wir das Zeichen, löschen oder fügen hinzu.

In diesem Kapitel lernen wir die Variablen kennen. Wir können sie mit Schachteln vergleichen, auf denen ein bestimmter Name steht. Das Arbeiten mit Variablen wird als erstes an einem Beispiel deutlich: die Berechnung des Inhalts eines Zylinders. Das Kapitel endet mit der Behandlung sogenannter Standardfunktionen. Sie lassen sich voll und ganz mit den Funktionstasten eines wissenschaftlichen Taschenrechners vergleichen.



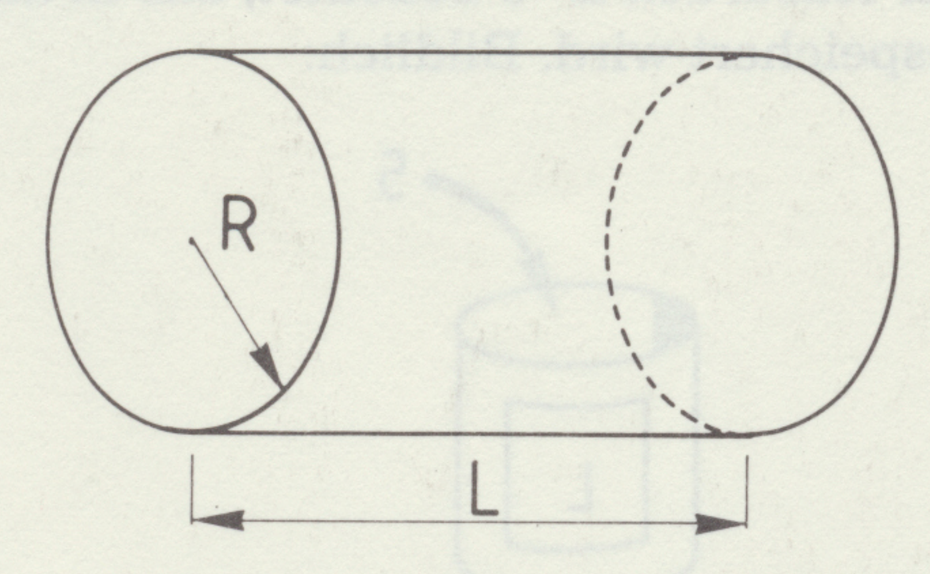
6. Variablen und noch mehr über Rechnen

Einführung

Die bisher behandelten Programme sind nicht sehr anregend. Computer sind aber als wahre Rechenwunder bekannt. Wir können noch deutlicher sein: ohne Computer wären viele Wunder unseres Jahrhunderts gar nicht möglich. Man denke etwa an die Weltraumflüge. Großbetriebe, Universitäten, Ingenieursbüros können ohne Computer heutzutage nicht mehr auskommen. Wir richten daher unsere Aufmerksamkeit jetzt durchaus berechtigt voll auf rechnerische Programme.
Das Rechnen ist bei vielen Leuten nicht gerade ein beliebtes Thema. Wir wollen aus diesem Grunde möglichst einfach bleiben und uns anschließend mit einem Programm befassen, mit dem wir den Inhalt eines Zylinders berechnen. Das erste Programm wird sehr einfach aussehen. Danach werden wir sehen, wie wir dieses Programm für allgemeine Verwendung einsetzen können.

Der Inhalt eines Stabes

Zur Inhaltsberechnung eines Zylinders müssen wir die kreisfläche ermitteln und diesen Wert mit der Länge multiplizieren. Bildlich dargestellt:



Oberfläche des Querschnitts = 3,14159xRxR
Zylinderinhalt = Oberfläche des Querschnitts x L = 3,14159xRxRxL

Wir sehen, daß der Inhalt eines Zylinders im allgemeinen von zwei Größen abhängt: dem Radius R und der Länge L.

Das folgende Programm berechnet den Inhalt eines Zylinders mit den Werten:

L=5
R=7

Programm:

```
10 L=5
20 R=7
30 PI=3.14159
40 UMFANG=PI*R*R
50 I=UMFANG*L
60 PRINT I
```

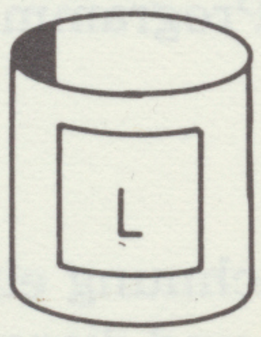
Nach dem RUN-Befehl erscheint:

```
769.68955
```

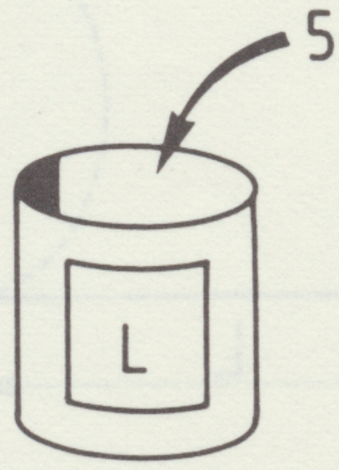
Wir werden dieses Programm jetzt Zeile für Zeile behandeln. In der Zeile 10 sieht man den Ausdruck:

L=5

Dieser Ausdruck bewirkt, daß der Computer einen Teil des Speichers reserviert und hier gewissermaßen das Etikett L aufklebt. Diesen Teil des Speichers können wir als eine Schachtel ansehen. Bildlich:

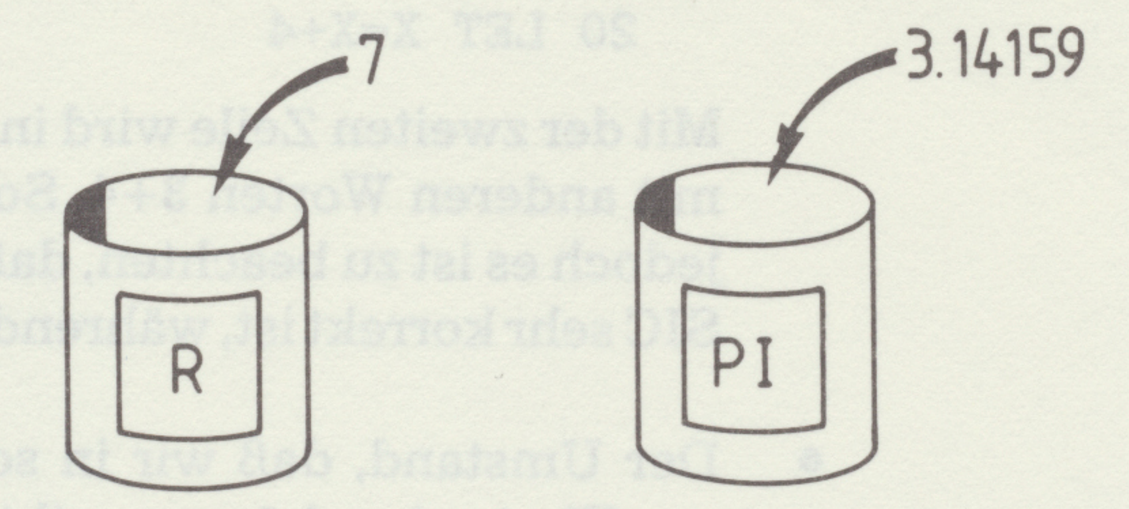


Der Ausdruck L=5 bedeutet, daß in die Schachtel L die Zahl 5 gespeichert wird. Bildlich:



Wenn nun im Programm auf L hingewiesen wird (Zeile 50), dann schaut sich der Computer gewissermaßen den Inhalt dieser Schachtel an.

Die Zeilen 20 und 30 können wir ebenso ins Bild bringen:



Achten Sie darauf, daß in PI die Zahl 3.14159 gespeichert wird und nicht 3,14159, denn Kommas, die wir üblicherweise bei Bruchzahlen verwenden, werden bei MSX-BASIC grundsätzlich durch einen Punkt ersetzt. Auch beim Resultat dieses Programmes sehen wir den 'Dezimalpunkt'.
Wer einige mathematische Kenntnisse hat, erkennt in der Zahl 3.14159 die Zahl π (Aussprache: pi).
Zeile 40 zeigt eine Bearbeitung: in der Schachtel mit dem Etikett UMFANG ist der Wert enthalten, den man erhält, wenn PI mit RxR multipliziert wird.
Unter UMFANG wird also der Wert 3.14159x7x7 gespeichert.
Der so ermittelte Wert wird in Zeile 50 gebraucht, um den schließlichen Wert des Inhalts zu bestimmen. Dieser wird in der Schachtel mit dem Etikett I untergebracht.
Schließlich wenden wir eine PRINT-Anweisung an, um den Inhalt der letzten Schachtel abzubilden:

```
PRINT I
```

abbilden *Inhalt der Schachtel mit Etikett I*

Sie sehen, daß hier nicht der Buchstabe I abgebildet wird, denn dann hätte die Anweisung lauten müssen PRINT "I", d.h. wir hätten dazu das I zwischen Anführungszeichen setzen müssen.

Zu diesem ersten Programm wollen wir folgendes anmerken:

- Wir hätten an Stelle von

  ```
  10 L=5
  ```

  auch tippen können:

  ```
  10 LET L=5
  ```

  Wörtlich würde das heißen: 'L soll ebenso groß sein wie 5' oder besser 'L wird 5'.
  Mit diesem Ausdruck betonen wir, daß der Computer eine Handlung ausführt: es wird etwas in die Schachtel mit dem Etikett L eingegeben. Diese Handlung wird bei folgendem Beispiel deutlich:

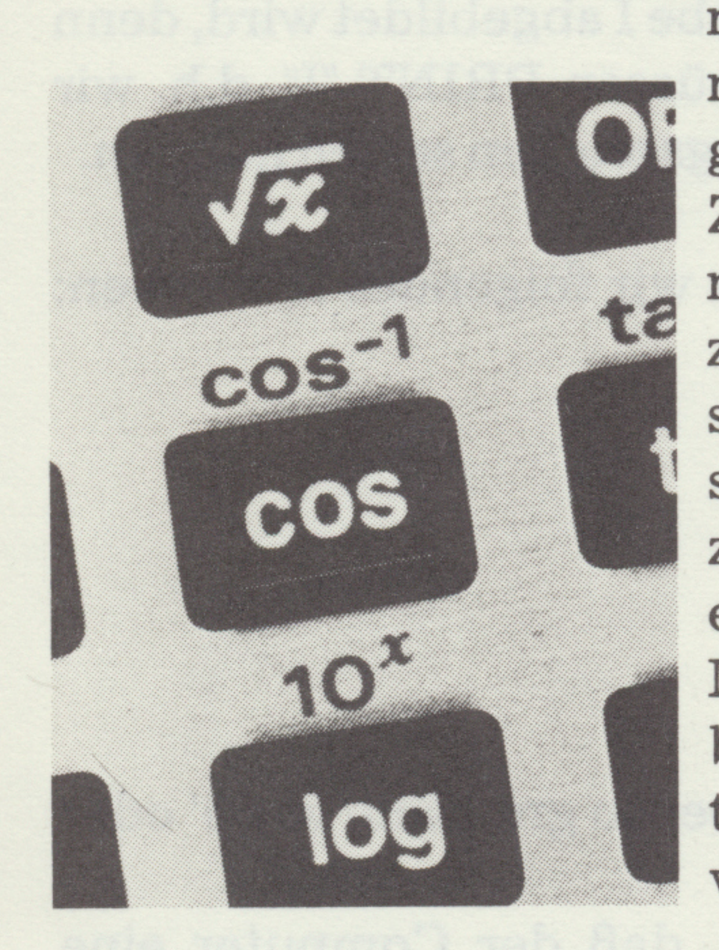
```
10 LET X=3
20 LET X=X+4
```

Mit der zweiten Zeile wird in X der alte Wert von X eingegeben, mit anderen Worten 3+4. So enthält X schließlich den Wert 7, jedoch es ist zu beachten, daß ein Ausdruck wie X=X+4 in BASIC sehr korrekt ist, während er mathematisch nicht erlaubt ist.

- Der Umstand, daß wir in solch eine Schachtel einen beliebigen Wert geben können, gibt an, daß der Inhalt variabel ist. Aus diesem Grund sprechen wir in der Regel von einer Variablen statt von einer Schachtel. Unser Programm hat also die Variablen L, R, PI, UMFANG und I.
  Wir werden daher auch in der Folge über die 'Wertzuweisung einer Variablen' sprechen und nicht vom 'Eingeben eines Wertes in eine Schachtel'.

- Zur Bezeichnung von Variablen gelten bestimmte Regeln. Diese sind:
  - Nur die beiden ersten Buchstaben einer Bezeichnung werden vom Computer als Zuweisung erkannt. So sind die Namen EINS und EINZIG für den Computer gleich, weil er nur EI erkennt.
  - Wir dürfen keine reservierten Wörter als Namen benutzen, beispielsweise nicht IF, weil diese Bezeichnung reserviert ist. Die reservierten Namen findet man im Anhang.

Standardfunktionen



Das gezeigte Beispiel war noch recht einfach. Wir können uns kaum vorstellen, daß wir auf diese Weise komplizierte Berechnungen durchführen können. Mit der Behandlung der sogenannten Standardfunktionen wird die Einsicht wesentlich vergrößert, was wir mit MSX BASIC berechnen können.
Zur Erklärung der Standardfunktionen machen wir zuerst einmal einen kurzen Seitensprung zum Taschenrechner, und zwar zum Rechner mit mathematischen Funktionen. Wenn wir auf solch einem Rechner die Wurzel aus einer Zahl berechnen müssen, tippen wir die Zahl und danach auf die Taste mit dem Wurzelzeichen. So berechnet man die erwünschte Funktion durch einfache Betätigung der entsprechenden Taste.
Mit MSX-BASIC können wir ebenso einfach Funktionswerte berechnen. Diese Funktion rufen wir auf durch einen bestimmten Ausdruck und Einklammerung der Zahl, deren Funktionswert berechnet werden soll. Also:

Ausdruck (Wert)

↗ *Funktionsangabe* ↖ *Angabe des Wertes, dessen Funktion zu bestimmen ist.*

Beispiele:

```
10 A=SQR(4)
20 PRINT A
```

Resultat:

```
2
```

In Zeile 10 wird A die Quadratwurzel aus 4 zugewiesen. Der Ausdruck, der in diesem Beispiel die Funktion kennzeichnet, ist gleich SQR. Eingeklammert ist der Wert, dessen Wurzel gezogen werden soll, in diesem Fall (der Einfachheit halber) 4. Hierzu wollen wir noch anmerken, daß zwischen Klammern auch eine Variable stehen darf oder auch ein mathematischer Begriff.
Beispiele:

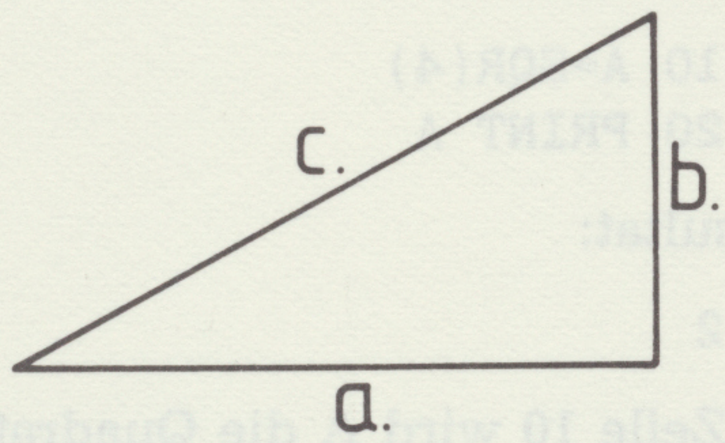
`B=SQR(A)` : Hier ist der Wert mit der Variablen A angegeben.

`C=SQR(SQR(5)+4+A)` : Hier wird der Wert mittels SQR(5)+4+A angegeben. Dieser Begriff enthält selbst auch wieder eine Funktion.

In der nachstehenden Tabelle werden die bekanntesten Funktionen gezeigt. Anhang E gibt eine Übersicht aller Funktionen.

MSX BASIC Bezeichnung	*Algebraische Notierung*	*Bedeutung*
ABS(X)	|x|	Absoluter Wert von X. Dies ist der Wert 'ohne Zeichen'.
ATN(X)	arctan(x)	Arcustangens von X. Ergebnis zwischen -pi/2 und pi/2.
COS(X)	cos (x)	Cosinus von X:X im Bogenmaß
EXP(X)	e^x	Exponentialfunktion von X.
INT(X)	keine	Abrundung von X immer nach unten. So ergibt INT(3.8) den Wert 3 und INT (-3.1) den Wert -4. Eine 'echte' Abrundung ist möglich mit INT(X+0,5)
LOG(X)	^{e}log(x)	Ergibt den sog. natürlichen Logarithmus von X.
SGN(X)	keine	Ergibt den Wert -1, 0 oder 1, je nachdem ob X negativ, 0 oder positiv ist.
SIN(X)	sin(x)	Sinusfunktion von X:X im Bogenmaß
SQR(X)	$\sqrt{x}$	Quadratwurzel aus X
TAN(X)	tan(x)	Tangensfunktion von X im Bogenmaß.

Wir bemerken, daß Potenzieren in dieser Tabelle nicht enthalten ist. Dafür verwenden wir das Zeichen ∧. So ergibt PRINT 2∧3 den Wert 8 (=2^3).
Schließlich noch ein einfaches Beispiel des Arbeitens mit Funktionen. Folgendes Programm berechnet die Hypotenuse eines rechtwinkligen Dreiecks.



Programm:

```
10 INPUT A
20 INPUT B
30 Z2=A∧2+B∧2
40 C=SQR(Z2)
50 PRINT C
```

Wir überlassen es dem Leser, dieses Programm praktisch zu erproben.

Mehrere Anweisungen in einer Zeile

MSX BASIC bietet uns die Möglichkeit, mehrere Anweisungen in eine Zeile zu setzen.
Wir müssen das Zeichen : als Trennungszeichen anwenden. Dies illustriert das folgende Programm, das angibt, wie wir einen Wertaustausch der Variablen A und B vornehmen können.
Ohne Trennungszeichen:

```
10 A=10
20 B=5
30 H=B
40 B=A
50 A=H
60 PRINT A
70 PRINT B
```

Resultat:

```
5
10
```

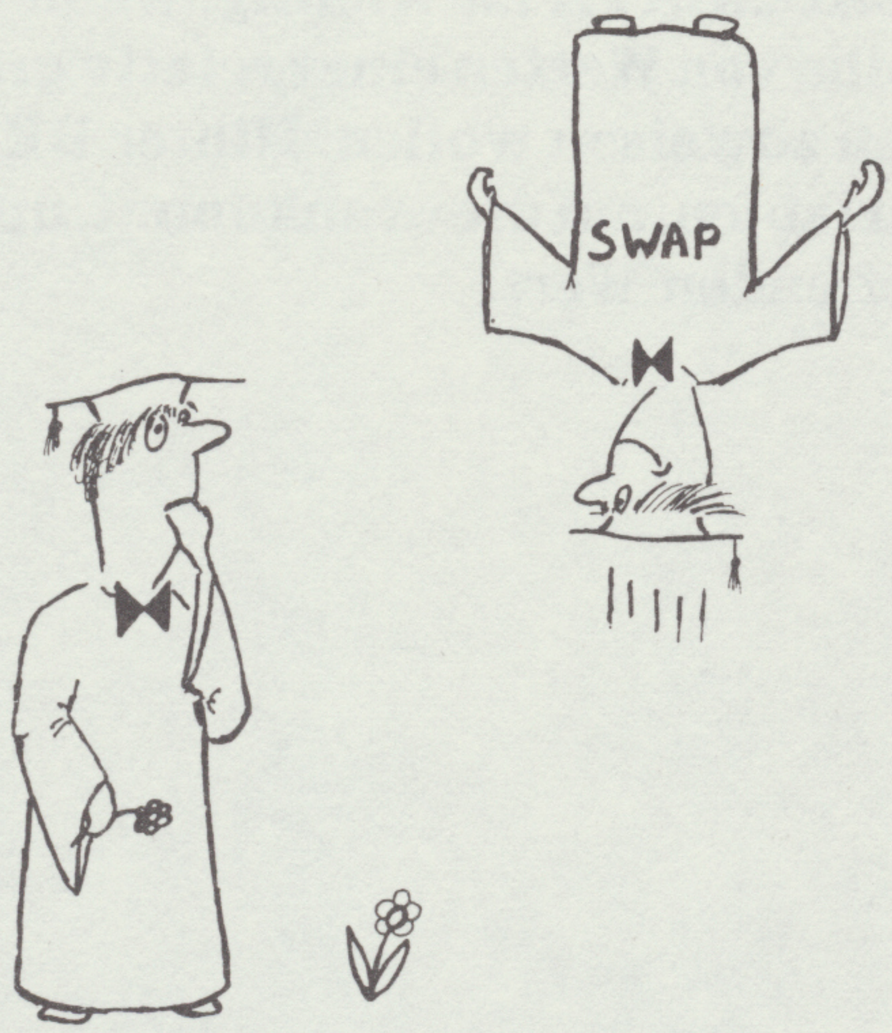
Mit Trennungszeichen:

```
10 A=10:B=5:H=B:B=A:A=H
20 PRINT A:PRINT B
```

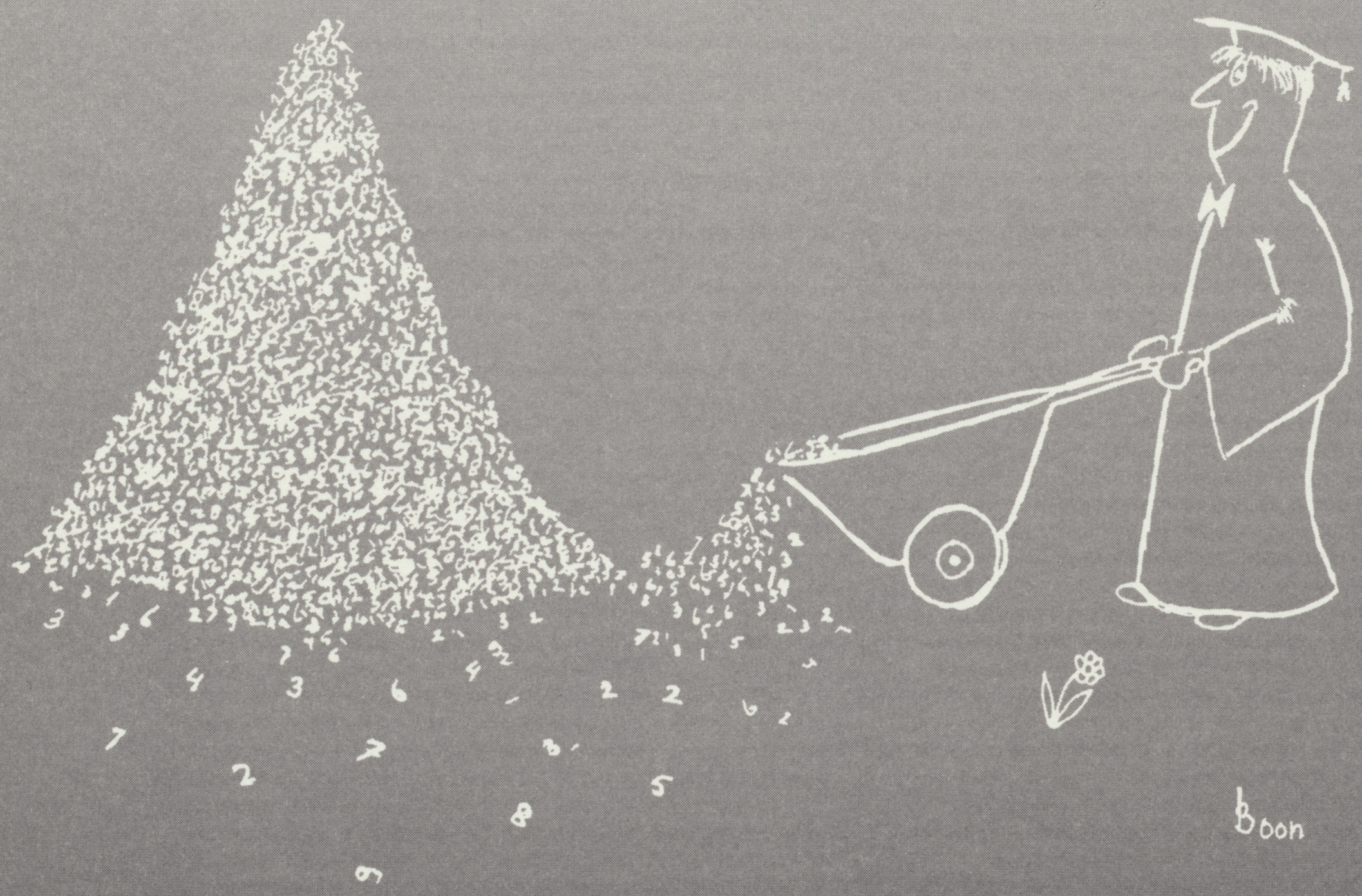
MSX BASIC hat übrigens eine besondere Anweisung zur Durchführung dieses 'Wechseltricks'. Wenn wir die Anweisung geben:

```
SWAP A,B
```

so werden die Werte von A und B vertauscht.



In diesem Kapitel erweitern wir unser Programmierwerkzeug mit den Anweisungen INPUT und READ...DATA. INPUT ist eine Anweisung, mit der wir einer Variablen einen Wert zuweisen können, ohne dabei das Programm zu verändern. Dies ist sehr wichtig, weil wir auf diese Weise Programme für einen allgemeineren Bereich entwickeln können. Die Form READ...DATA ist wichtig, wenn wir eine relativ große Reihe von Werten einer relativ großen Reihe von Variablen zuweisen wollen. Hinter READ kommen dann die Variablen, auf die es ankommt, und hinter DATA die betreffenden Werte.



7. INPUT, READ und DATA

INPUT Als erstes behandeln wir jetzt die INPUT-Anweisung. Sie bietet uns die Möglichkeit, Variablen während der Ausführung eines Programmes einen Wert zuzuweisen. Noch wesentlicher ist es vielleicht, daß wir mit dieser Anweisung Programme für allgemeinere Anwendung aufstellen können. Wir wollen hier noch einmal das Programm des vorigen Kapitels betrachten:

```
10 L=5
20 R=7
30 PI=3.14159
40 UMFANG=PI*R*R
50 I=UMFANG*L
60 PRINT I
```

Statt dieses Programmes hätten wir natürlich auch eingeben können:

```
PRINT 3.14159*7*7*5
```

um zu dem gleichen Resultat zu kommen.
Wir haben das nicht getan, um aufzuzeigen, wie wir ein Rechenschema in den Computer einspeichern können, wobei zur Bestimmung des Resultates nur noch die Werte R und L eingesetzt werden müssen.
Um beispielsweise den Inhalt eines Zylinders mit L=34 und R=10 bestimmen zu können, müssen wir nun die Zeilen 10 und 20 verändern in

```
10 L=34
20 R=10
```

und nach dem RUN-Kommando erscheint die verlangte Antwort. Doch die Veränderung eines Programmes zur Durchführung einer Berechnung ist auch nicht besonders empfehlenswert. Dabei könnten wir schließlich Fehler machen.
Viel besser würde jedoch das Programm, wenn nach dem RUN-Befehl auf dem Bildschirm Mitteilungen erschienen, die ange-

ben, daß die entsprechenden Werte für L und R eingegeben werden müssen.
Diese Möglichkeit wird uns in Form der INPUT- Anweisung geboten. Sie sieht folgendermaßen aus:

INPUT 'wiederzugebender Text'; Name Variable

Beispiel:

```
INPUT "LÄNGE EINGEBEN"; L
```

Das Resultat dieser Anweisung ist, daß der Computer den Text abbildet, dahinter ein Fragezeichen, und gleich danach unterbricht er das Programm. Der Anwender muß nun erst eine Zahl eingeben, beispielsweise:
5 (anschließend RETURN)
Sofort nach Betätigung der RETURN-Taste wird dieser Zahlenwert der Variablen L zugewiesen, und der Computer setzt die Programmausführung fort.
Unser Programm sieht nun mit den INPUT-Anweisungen folgendermaßen aus:

```
10 INPUT "LÄNGE EINGEBEN"; L
20 INPUT "RADIUS EINGEBEN"; R
30 PI=3.14159
40 UMFANG=PI*R*R
50 I=UMFANG*L
60 PRINT "INHALT="; I
```

Ein Beispiel eines Ergebnisses ist:

```
LÄNGE EINGEBEN? 5
RADIUS EINGEBEN? 7
INHALT=769.68955
```

Wir sehen, daß zur PRINT-Anweisung in diesem Programm zugleich auch Text aufgenommen wurde, wobei dieser Text in Anführungszeichen kam und ; (Semikolon) als Trennungszeichen dient.

Das Programm ist im Vergleich zu unserem ersten Programm wesentlich verbessert. Die Struktur liegt geradezu völlig fest. Im Programm selbst brauchen wir keine Veränderungen mehr vorzunehmen, wenn wir den Inhalt eines beliebigen Zylinders berechnen wollen.

Anmerkungen: Zur INPUT-Anweisung ist noch folgendes anzumerken:

- Wir dürfen mehrere Variablen in eine INPUT-Anweisung aufnehmen, z.B.:

```
INPUT "EINGABE VON A, B UND C:"; A, B, C
```

Nach Unterbrechung des Programmes bringt der Computer den Text nebst Fragezeichen, und wir müssen nun die drei Zahlen, getrennt durch Kommas, eingeben.

- Den Text der INPUT-Anweisung dürfen wir auch weglassen, z.B.:

```
INPUT A
```

Nach der Unterbrechung zeigt der Computer nun nur noch ein Fragezeichen, und wir müssen den Zahlenwert eingeben.

Aufbau des Programmes

An dieser Stelle wollen wir uns mit dem Aufbau des Programmes befassen. Ausgangspunkt war eine Formel, bei der im voraus einigen Größen ein bestimmter Wert zuzuweisen war. Wir sahen, daß hierzu folgender Aufbau gewählt wurde:

Verlangte Werte eingeben
Berechnung nach Formel ausführen
Resultat wiedergeben

Eine derartige Gestaltung finden wir immer dann, wenn es sich um Berechnungen nach Formeln handelt.

READ und DATA

Bisher haben wir zwei Möglichkeiten besprochen, wie wir Variablen Werte zuweisen können.
Die erste Methode ging aus von der Wertzuweisung über die LET-Anweisung, etwa:

```
LET L=3
```

oder kürzer

```
L=3
```

Bei der zweiten Möglichkeit wurde von der INPUT- Anweisung ausgegangen, z.B.:

```
INPUT L
```

Die dritte Methode, die wir jetzt besprechen wollen, geht von dem Wunsch aus, einer relativ großen Anzahl von Variablen Werte zuzuweisen. Hierbei gilt, daß diese Werte in der Regel nicht von Programm zu Programm zu verändern sind.
In einer derartigen Situation könnten wir natürlich auch eine ganze Reihe von LET-Anweisungen aufnehmen, aber die Mög-

lichkeit mit READ und DATA ist, wie wir sehen werden, sehr viel eleganter. Hierfür gleich ein Beispiel:

```
10 READ A,B,C,D
20 F=A+B+C+D
30 PRINT F
40 DATA 3,7,4,8
```

Die Folge der READ-Anweisung in Zeile 10 bedeutet, daß der Computer 'weiß', daß eine Reihe von Variablen genannt wird. Diese Werte werden in der Zeile genannt, die mit DATA beginnt.
So wird A der Wert 3, B der Wert 7, C der Wert 4 und D der Wert 8 zugewiesen.
Das Resultat des Programmes bestätigt dies:

```
22
```

Es ist zu beachten, daß die DATA-Zeile an sich keine ausführbare Anweisung enthält. Sie ist einfach eine Art Notizzettel, auf dem der Computer die gesuchten Werte findet. Wir hätten die READ-Liste auch aufteilen können:

```
10 READ A,B
15 READ C,D
```

und ebenso hätten wir die DATA-Liste aufteilen können:

```
40 DATA 3,7,4
50 DATA 8
```

Für den Computer macht dies alles keinen Unterschied. Er liest diese Daten, als handele es sich um eine fortlaufende Liste. Diesen Mechanismus kann man sich am besten so vorstellen, als kennzeichne der Computer den ersten Wert der DATA-Liste mit einem Pfeil. Jedesmal, wenn er nun mit einer READ-Anweisung einer Variablen einen Wert zugewiesen hat, schiebt sich der Pfeil eine Stelle weiter:

```
40 DATA 3,7,4,8
          ↑
```

Pfeil verschiebt sich um jeweils eine Stelle

Sobald der letzte Wert der DATA-Liste erreicht ist, verschiebt der Computer den Pfeil zum ersten Wert der nächsten DATA-Liste, sofern eine vorhanden ist.
Diesen Pfeil können wir auch mit der folgenden Anweisung in seine Ausgangsposition zurückversetzen:

```
RESTORE
```

Beispiel:

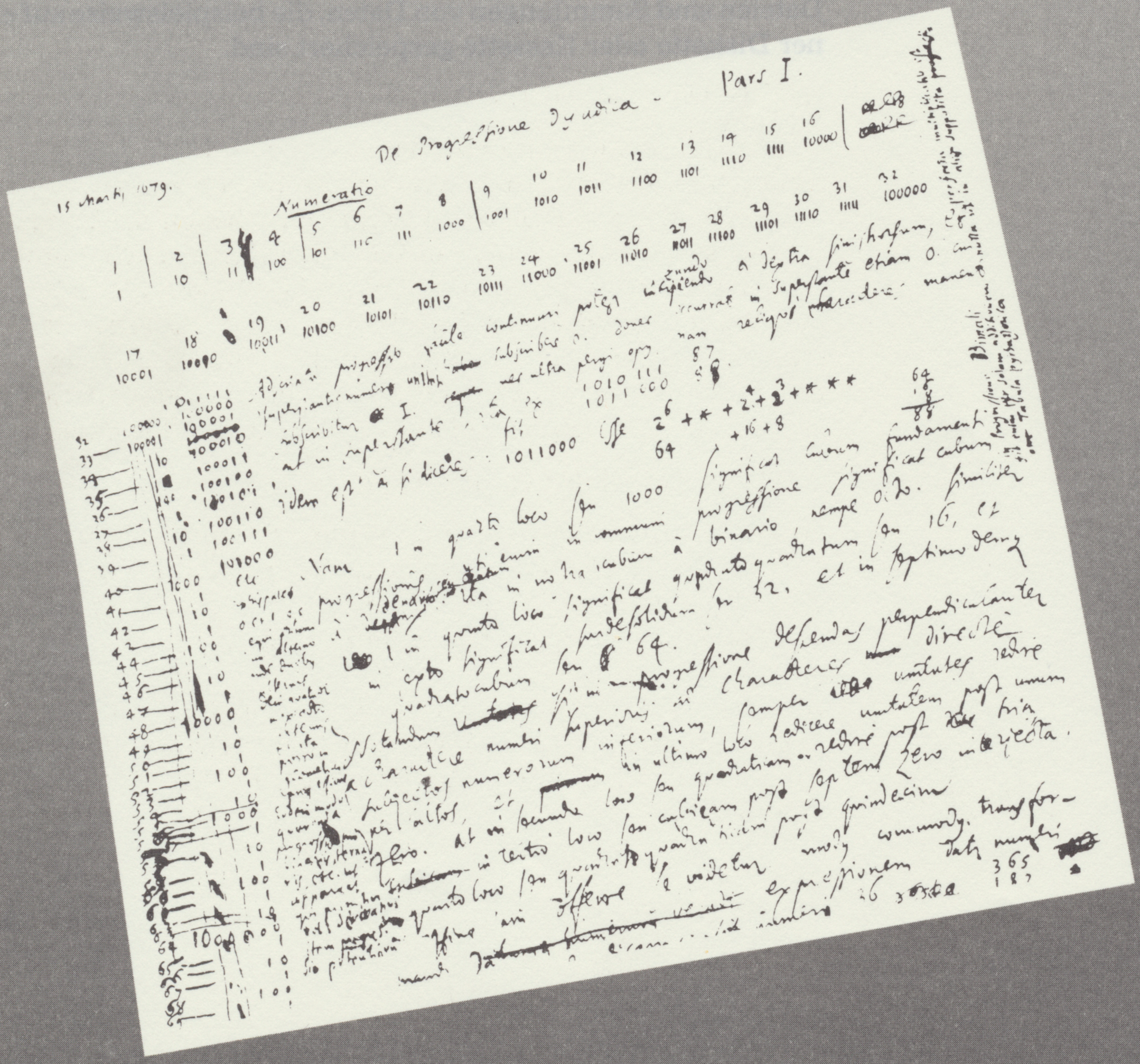
```
10 READ A,B,
20 RESTORE
30 READ C,D,
40 F=A+B+C+D
50 PRINT F
60 DATA 3,7,4,8
```

Resultat:

```
20
```

Beachten Sie, daß das Ergebnis jetzt 20 ist. Dies ist eine Folge der RESTORE-Anweisung; A und B wird wie auch C und D nun 3 und 7 zugewiesen.
Sind nun mit der Besprechung von LET, INPUT, READ und DATA alle Möglichkeit zur Zuweisung von Daten behandelt? Die Antwort ist Nein!
Das wäre nämlich auch gar nicht praktisch, wenn wir an die vielen verwaltungstechnischen Einsatzmöglichkeiten von BASIC denken.
Die letzte Möglichkeit, auf die wir in diesem Augenblick nicht weiter eingehen wollen, geht aus von der Eingabe von Werten aus Dateien.
Dateien sind Sammlungen von Daten, die beispielsweise auf einer Diskette oder Kassette gespeichert sind.

In diesem Kapitel gehen wir auf verschiedene Zahlen tiefer ein. Als erstes betrachten wir, wie man sehr große und auch sehr kleine Zahlen in sogenannter wissenschaftlicher Notierung wiedergeben kann.
Anschließend wird auf die Begriffe doppelte und einfache Genauigkeit eingegangen. Hierbei handelt es sich um die Genauigkeit, mit der Zahlen in einen Computer eingespeichert werden können.
Schließlich geben wir eine Einführung zu binären, oktalen und hexadezimalen Zahlen. Es handelt sich hierbei um Notierungsformen, bei der die Position einer Zahl mit einer bestimmten Potenz übereinstimmt.



8. Zahlen, groß und klein und in allerlei Gestalt

Wissenschaftliche Notierung

Wir sind bisher bereits verschiedenen Arten von Zahlen begegnet, nämlich ganzen Zahlen, die auch als Integer bezeichnet werden, sowie den Zahlen mit einem dezimalen Punkt.
Jetzt wollen wir eine neue Form besprechen, in der wir Zahlen mit einem dezimalen Punkt darstellen können. Diese Form ist vor allem geeignet, wenn es sich um sehr große oder sehr kleine Zahlen handelt. Derartige Zahlen kommen beispielsweise in physikalischen oder astronomischen Berechnungen vor.
Um dies etwas greifbarer zu machen, wollen wir einmal an die elektrische Ladung eines Elektrons denken.
Diese Ladung, ausgedrückt in der Einheit Coulomb, ist:

0.000 000 000 000 000 000160219

Dies ist tatsächlich ein besonders kleiner Wert. Wir haben absichtlich Zwischenräume zwischen die Nullen gesetzt, um sie leichter zählen zu können (bei Computern dürfen wir übrigens nie Zwischenräume zwischen Zahlen einsetzen). Wie können wir eine derartige Zahl nun auf einfache Weise notieren?
Die Antwort lautet: stelle diese Zahl als eine Multiplikation von zwei Zahlen dar. Die erste Zahl gibt dann die 'wirklichen' Ziffern in der ursprünglichen Zahl an und die zweite sofort die Anzahl Nullen, die hier im Spiel sind. Obenstehendes Beispiel:

1.60219 x Anzahl der vorhandenen Nullen

Bei der Zahl, die angeben muß, wieviele Nullen in der ursprünglichen Zahl vorkommen, wird eine sogenannte Potenz der Zahl 10 genommen. In der Tabelle auf der nächsten Seite bringen wir einige Beispiele der Potenzen von 10.

Aus der Tabelle können wir ersehen, daß beispielsweise 10^4 übereinstimmt mit 10000. Mit anderen Worten: die Anzahl der Nullen stimmt hier direkt überein mit der Potenz (Ziffer rechts über der 10).

Eine Seite aus dem Notizbuch von Leibnitz, auf der er die binären Zahlen einführt.

Wörtlich	*Notierung*	*Bedeutung*	*Wert*
10 in der 1. Potenz	10^1	10	10
10 in der 2. Potenz	10^2	10x10	100
10 in der 3. Potenz	10^3	10x10x10	1000
10 in der 4. Potenz	10^4	10x10x10x10	10000
10 in der -1. Potenz	10^{-1}	1/10	0.1
10 in der -2. Potenz	10^{-2}	1/(10x10)	0.01
10 in der -3. Potenz	10^{-3}	1/(10x10x10)	0.001
10 in der 0. Potenz	10^0	10/10	1

Beispiel:
Mit welcher Potenz stimmt 1000 000 000 überein?
Antwort: Anzahl der Nullen zählen; es sind 9, und damit erhalten wir 10^9.

Aus der Tabelle ist ebenfalls ersichtlich, daß 0.001 übereinstimmt mit 10^{-3}. Auch bei Zahlen, die kleiner sind als 1, kann man offenbar die Nullen zählen, um die Potenz zu bestimmen.

Beispiel:
Mit welcher Potenz stimmt die nachstehende Zahl überein?

0.000 000 000 000 000 000 1

Antwort: Anzahl Nullen zählen; es sind 19, und so kommen wir auf 10^{-19}.
Dies war übrigens genau die Anzahl der Nullen, die in der Zahl vorkam, die die Ladung des Elektrons angab. Wir können also diese Ladung auch folgendermaßen andeuten:

1.60219×10^{-19}

Die erste Zahl wird Mantisse genannt, und die zweite Zahl gibt den Exponenten an.
Beim Computer geben wir die obige Zahl in folgender Notierung an:

1.60219E-19

Wir gebrauchen also den Buchstaben E, um die Mantisse vom Exponenten zu unterscheiden.
Um einige Erfahrungen mit dieser Notierung zu machen, kann man das folgende Kurzprogramm benutzen:

```
10 INPUT A
20 INPUT B
30 C=A*B
40 PRINT C
```

Beispiele:

A=987654	B=456789	ist: 451149483006
A=98765434	B=987654321	ist: 9.7546105778997E+16'
A=.0000000008	B=.0000000007	ist: 5.6E-19
A=8000000000	B=7000000000	ist: 5.6E+19

Doppelte und einfache Genauigkeit

Zur Wiedergabe von Zahlen benutzt unser Computer eine beschränkte Anzahl von Speicherregistern (Wörter). Dies bedeutet auch, daß die Berechnungen eine beschränkte Genauigkeit haben. MSX-BASIC bietet dem Anwender die Möglichkeit zur Arbeit sowohl mit der sogenannten doppelten Genauigkeit als auch mit einfacher Genauigkeit. Bei doppelter Genauigkeit werden immer 8 Register (Bytes) zur Speicherung von Zahlen benutzt, bei einfacher Genauigkeit nur 4. Standardmäßig arbeitet unser Computer immer mit Zahlen doppelter Genauigkeit. Wollen wir mit einfacher Genauigkeit arbeiten, dann setzen wir immer ein Ausrufezeichen (!) hinter die Zahl.

Die Variablen, denen wir Zahlen mit einfacher Genauigkeit zuweisen, unterscheiden sich von Variablen mit doppelter Genauigkeit durch Eingabe des Ausrufezeichens (!) direkt hinter der Variablen. Die folgenden Beispiele zeigen den Unterschied zwischen Berechnungen mit einfacher und doppelter Genauigkeit.

```
10 A=10/3                10 A!=10!/3!
20 PRINT A               20 PRINT A!
```

Resultat	Resultat
3.3333 333 333 333 3	3.333333

Hierzu noch folgende Anmerkungen:

- Bei Zahlen mit doppelter Genauigkeit dürfen wir ans Ende der Zahl das Zeichen # setzen. Ebenso dürfen wir dieses Zeichen als letztes Zeichen einer Variablen eingeben. So geben wir nachdrücklich an, daß wir mit doppelter Genauigkeit arbeiten.

- Wenn wir bei Zahlen mit doppelter Genauigkeit den Buchstaben D statt E gebrauchen (z.B. 23.45156321D39), so geben wir damit nachdrücklich an, daß wir mit doppelter Genauigkeit arbeiten.

- Es ist fortwährend daran zu denken, ob es wohl ratsam ist, Ergebnisse mit soviel Zahlen wiederzugeben. Wenn beispielsweise ein Zimmermann ein 10 Meter langes Brett in drei Teile zersägen muß, ist jedes Teil laut unserem Computer 3.333 333 333 333 3 m lang - aber kann unser Zimmermann so genau sägen?

- Die Anwendung von Zahlen mit einfacher Genauigkeit kann vor allem wichtig sein, wenn wir mit unserem Speicher sparsam umgehen wollen. Diese Speicherprobleme ergeben sich in der Praxis vor allem bei den noch zu besprechenden Arrays (Kapitel 11).

Ganze Zahlen

In verschiedenen Fällen wollen wir im Programm angeben, daß es sich um ganze Zahlen oder um 'Integer' handelt.
Bei der Wiedergabe von Zahlen in einem Programm ergeben sich hieraus keine Probleme: eine Zahl ohne Dezimalpunkt ist schließlich eine ganze Zahl. Variablen kann man jedoch nicht ansehen, ob sie nun eine ganze Zahl oder eine Zahl mit Dezimalpunkt darstellen. Um nun diesen Unterschied deutlich zu machen, setzt man ein %-Zeichen hinter den Namen.
Beispiele:

```
A%    B1%
```

Bitte beachten Sie, daß solch eine Variable immer nur eine ganze Zahl enthalten kann. Wenn das Ergebnis einer Berechnung keine ganze Zahl ist und ein derartiger Wert einer Integer-Variablen zugemessen wird, findet automatisch eine Abrundung (nach unten) statt.
Beispiel:

```
10 A%=20
20 B%=3
30 C%=A%/B%
40 PRINT C%
```

Resultat:

```
6
```

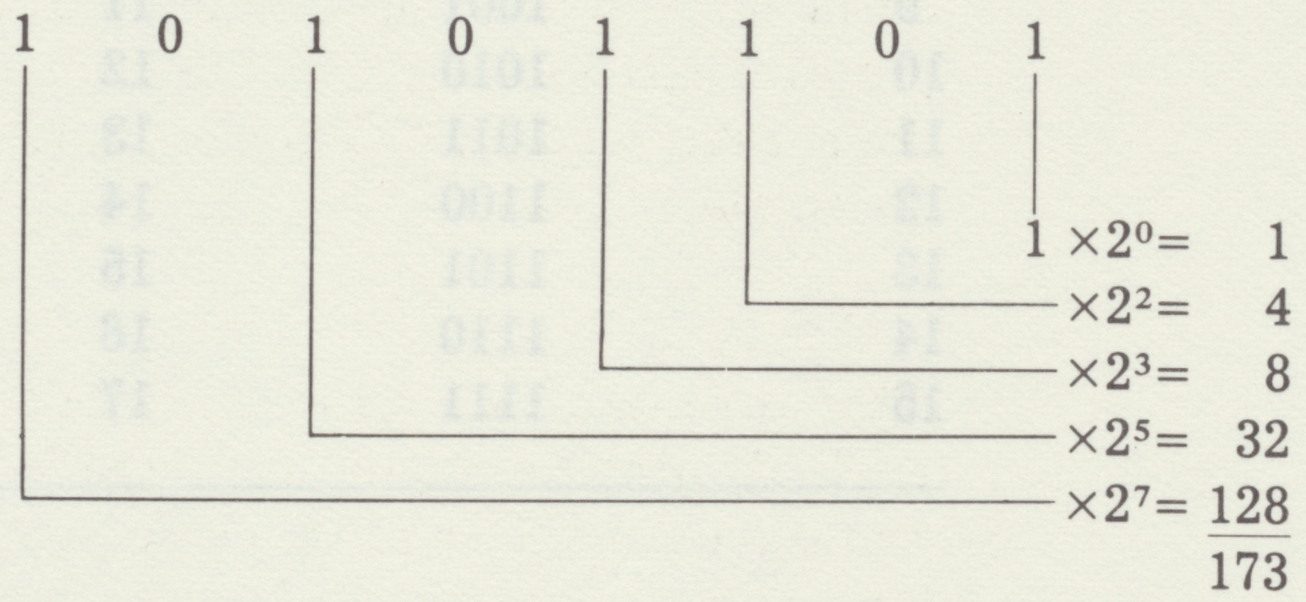
Binärzahlen

Zur Erklärung binärer Zahlen schauen wir uns zunächst einmal den Aufbau unserer 'gewöhnlichen', d.h. Dezimalzahlen an. Diese Zahlen sind immer aus Zehnerpotenzen aufgebaut (siehe Tabelle zu Beginn dieses Kapitels).
Beispiel:
Die Zahl 4736 ist folgendermaßen aus Zehnerpotenzen aufgebaut:

$$
\begin{array}{cccccccc}
4 & 7 & 3 & 6 & & & & \\
| & | & | & | & & & & \\
| & | & | & | & 6\times10^0=6\times & 1= & 6 \\
| & | & 3 & \text{——} & \times10^1=3\times & 10= & 30 \\
| & 7 & \text{——} & \text{——} & \times10^2=7\times & 100= & 700 \\
4 & \text{——} & \text{——} & \text{——} & \times10^3=4\times & 1000= & \underline{4000}+ \\
 & & & & & & 4736
\end{array}
$$

Wir sehen, wie hier jede Ziffer mit einer Zehnerpotenz übereinstimmt. Von rechts ausgehend zählen wir die nullte, erste, zweite und dritte Position, und diese Positionen stimmen dann genau mit den Potenzen von 10 überein.

Die binären Zahlen sind nur aus den Ziffern 0 und 1 zusammengesetzt, und die Position solch einer Ziffer stimmt wieder mit einer Potenz überein, und zwar mit einer Zweierpotenz.
Beispiel:
Mit welcher Dezimalzahl stimmt die Binärzahl 10101101 überein?
Ausarbeitung:



Bei MSX-BASIC können wir Binärzahlen auch direkt angeben, wenn wir &B vor die Zahl setzen.

```
10 A%=&B10101101
20 PRINT A%
```

Resultat:

```
173
```

Oktale und hexadezimale Zahlen

Nach dem Obigen können wir einfach angeben, was Oktal- und Hexadezimalzahlen sind.
Oktalzahlen sind Zahlen, deren Ziffer gemäß unserer Darlegung übereinstimmt mit einer Potenz von 8. Bei Hexadezimalzahlen stimmt die Position einer Ziffer mit einer Potenz von 16 überein.
Es ist eine interessante Feststellung, daß wir bei Dezimalzahlen genau 10 Ziffern verwenden (0 bis 9), um diese Zahlen darzustellen. Bei binären Zahlen gibt es nur zwei Ziffern (0 und 1) und bei den oktalen sind es natürlich 8 (0 bis 7).
Bisher keine Probleme. Bei Hexadezimalzahlen sind es natürlich 16 Ziffern - aber wir kennen nur die Ziffern von 0 bis 9. Wie geben wir aber dann die übrigen Ziffern bei Hexadezimalzahlen an? Nun, da verwenden wir die Buchstaben von A bis F.
Die folgende Tabelle zeigt die Dezimalzahlen 0-15, und diese werden zur Illustration zugleich auch in binärer, oktaler und hexadezimaler Form gezeigt.

Dezimal	*Binär*	*Oktal*	*Hexadezimal*
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

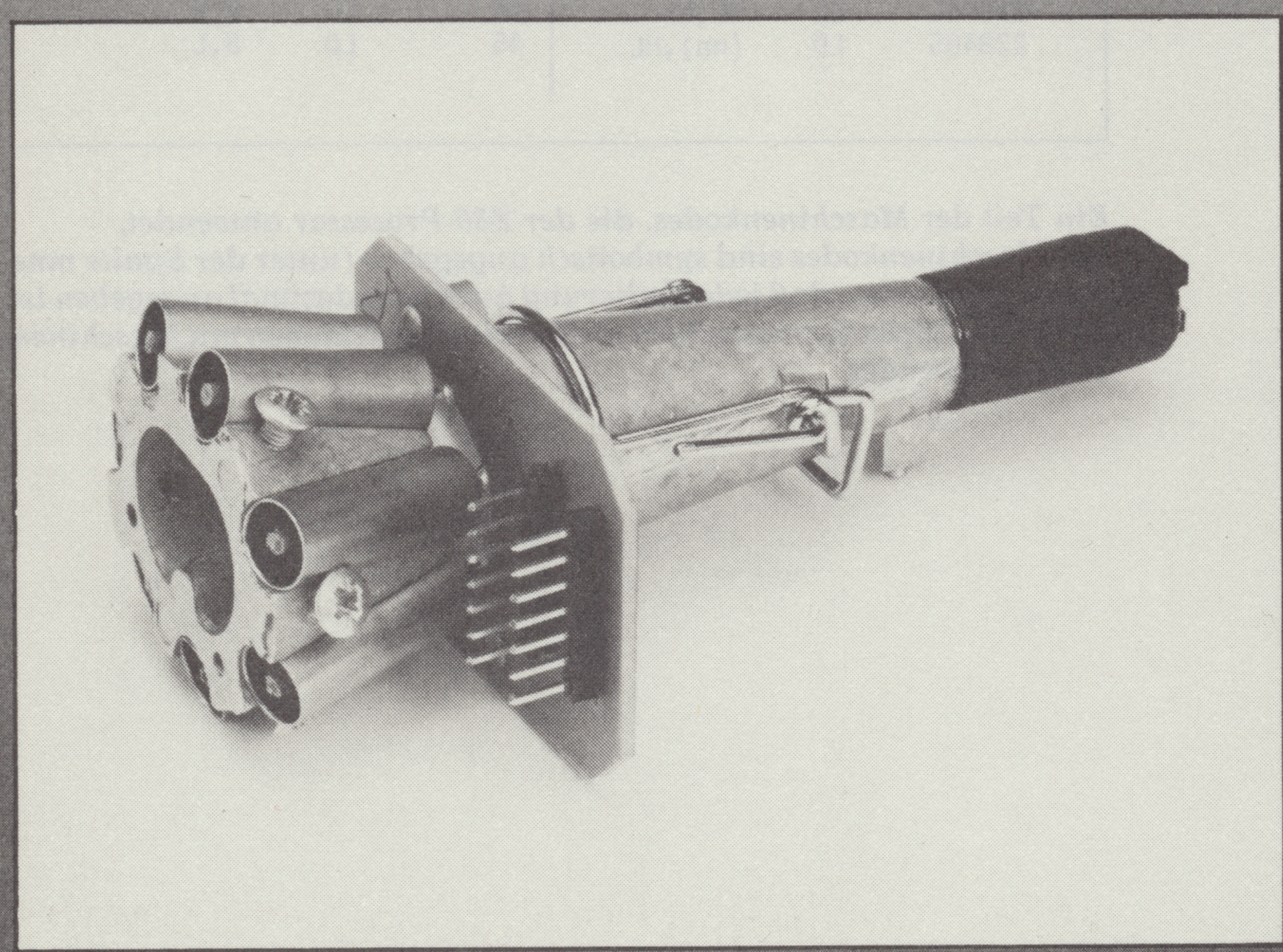
Die Nützlichkeit ist vorerst nicht zu erkennen. Doch wir werden noch sehen (u.a. in dem Kapitel über Sprites), welche Vorteile wir dadurch haben können.
Oktale und hexadezimale Zahlen dürfen als solche unmittelbar in das Programm aufgenommen werden. Vor eine oktale Zahl setzen wir &O und vor eine hexadezimale Zahl &H.
Beispiele:

```
PRINT &017 ergibt 15
PRINT &HF ergibt 15
PRINT &HFF ergibt 255
```

hex.kode	mnemonic		hex.kode	mnemonic	
00	NOP		23	INC	HL
018405	LD	BC,nn	24	INC	H
02	LD	(BC),A	25	DEC	H
03	INC	BC	2620	LD	H,n
04	INC	B	27	DAA	
05	DEC	B	282E	JR	Z,e
0620	LD	B,n	29	ADD	HL,HL
07	RLCA		2A8405	LD	HL,(nn)
08	EX	AF,AF'	2B	DEC	HL
09	ADD	HL,BC	2C	INC	L
0A	LD	A,(BC)	2D	DEC	L
0B	DEC	BC	2E20	LD	L,n
0C	INC	C	2F	CPL	
0D	DEC	C	302E	JR	NC,e
0E20	LD	C,n	318405	LD	SP,nn
0F	RRCA		328405	LD	(nn),A
102E	DJNZ	e	33	INC	SP
118405	LD	DE,nn	34	INC	(HL)
12	LD	(DE),A	35	DEC	(HL)
13	INC	DE	3620	LD	(HL),n
14	INC	D	37	SCF	
15	DEC	D	382E	JR	C,e
1620	LD	D,n	39	ADD	HL,SP
17	RLA		3A8405	LD	A,(nn)
182E	JR	e	3B	DEC	SP
19	ADD	HL,DE	3C	INC	A
1A	LD	A,(DE)	3D	DEC	A
1B	DEC	DE	3E20	LD	A,n
1C	INC	E	3F	CCF	
1D	DEC	E	40	LD	B,B
1E20	LD	E,n	41	LD	B.C
1F	RRA		42	LD	B,D
202E	JR	NZ,e	43	LD	B,E
218405	LD	HL,nn	44	LD	B,H
228405	LD	(nn),HL	45	LD	B,L

Ein Teil der Maschinenkodes, die der Z80-Prozessor anwendet. Die Maschinenkodes sind symbolisch angegeben (unter der Spalte mnemonic). Sie sehen, daß jede Kodierung auch hexadezimal angegeben ist. So werden Hexadezimalzahlen vielfach in Programmen mit Maschinenkodes verwendet.

In diesem Kapitel gehen wir ausführlich auf die PRINT-Anweisung ein. Als erstes zeigen wir, wie wir mehr zu druckende Dinge in eine PRINT-Anweisung aufnehmen können. Hierbei werden die Zeichen Komma(,) und Semikolon (;) als Trennungszeichen verwendet. Anschließend wird gezeigt, wie wir mit den Funktionen TAB und LOCATE die Position angeben können, an der etwas auf dem Bildschirm stehen muß. Die PRINT USING-Form ist wichtig, wenn es darauf ankommt, in welcher Form Zahlen dargestellt werden müssen. Die REM-Anweisung bietet schließlich die Möglichkeit, Kommentare in das Programm aufzunehmen.



9. PRINT, TAB, LOCATE, PRINT USING und REM

Trennungszeichen

In der Folge werden wir vor allem den Trennungszeichen in der PRINT-Anweisung besondere Aufmerksamkeit schenken.
Die PRINT-Anweisung ist sehr wichtig, weil der Druck der Ergebnisse, die wir auf Grund der PRINT-Anweisung regeln, schließlich eines der Themen ist, mit denen jeder Programmierer zu tun hat.
In den vorigen Kapiteln sahen wir, daß wir hinter die Anweisung PRINT verschiedene Aufträge eintippen können.
Wir illustrieren die verschiedenen Möglichkeiten wieder mit Beispielen.
Beispiel 1:

```
10 A=3
20 B=4
30 PRINT A*B
```

Resultat:

```
12
```

Dieses Beispiel zeigt auf einfache Weise, daß wir in eine PRINT-Anweisung nicht nur Namen der Variablen aufnehmen dürfen, sondern auch noch Bearbeitungen angeben dürfen.
Beispiel 2:

```
10 A=12
20 PRINT "A=";A
30 PRINT "A=",A
```

Resultat:

```
A=12
A=          12
```

Der Kopf eines Matrix-Druckers. In diesem Fall werden genau 7 Stifte angesteuert, die Symbole in Form von Punkten abdrucken.

Der Unterschied zwischen dem Resultat von Zeile 20 und von Zeile 30 liegt im Trennungszeichen.
Bei einem Semikolon steht das nächste abzudruckende Resultat direkt hinter dem vorigen. Gebraucht man ein Komma, so

gebraucht der Computer automatisch eine Einteilung in Spalten.
Bei dieser Spalteneinteilung ist immer ein Zwischenraum von 14 Positionen.
Man darf übrigens diese Trennungszeichen auch am Ende einer PRINT-Anweisung gebrauchen, aber schauen Sie sich einmal folgendes Beispiel an:

```
10 PRINT "ABC";"DEFG";
20 PRINT "H"; "IJ";"KLM"
```

Resultat:

```
ABCDEFGHIJKLM
```

Will man beim Drucken von Text eine Zeile auslassen, so gebraucht man eine PRINT-Anweisung ohne nähere Kennzeichnung.
Beispiel:

```
10 PRINT "A"
20 PRINT
30 PRINT "B"
```

Resultat:

```
A

B
```

Es ist deutlich, daß zwischen A und B jetzt eine Zeile ausgelassen ist.

TAB Mit der sogenannten TAB-Funktion können wir auch angeben, in welcher Spalte die abzudruckende Größe stehen soll.
Folgendes Beispiel verdeutlicht den Gebrauch von TAB:

```
10 PRINT TAB(1);"MSX"
20 PRINT TAB(3);"MSX"
30 PRINT TAB(5);"MSX"
```

Resultat:

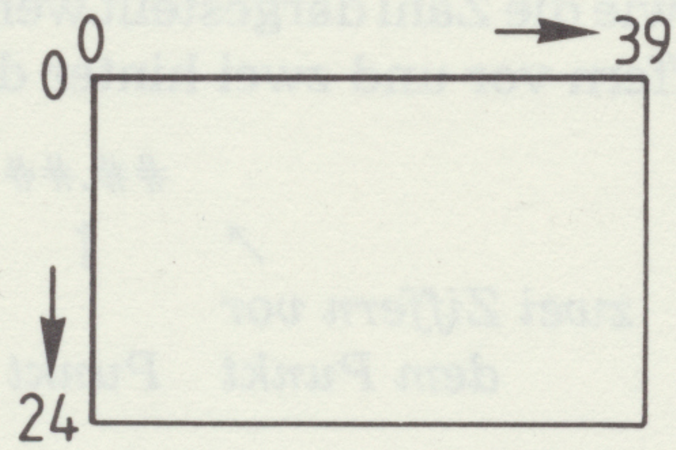
```
MSX
  MSX
    MSX
```

Wir sehen, daß hinter TAB eingeklammert stets die Spaltenposition angegeben ist, in der der Abdruck (in diesem Fall Text) stehen muß. Die Positionswerte gehen übrigens von 0-39.

LOCATE Die Funktion TAB bestimmt die Spaltenposition in der Zeile. LOCATE gibt den horizontalen wie den vertikalen Platz an. Es ist eine sehr kraftvolle Anweisung, mit der wir auf einfache Weise eine schönere Aufmachung eines abzudruckenden Textes erreichen können.
Die Form der LOCATE-Anweisung ist wie folgt:

LOCATE horizontale Position , vertikale Position

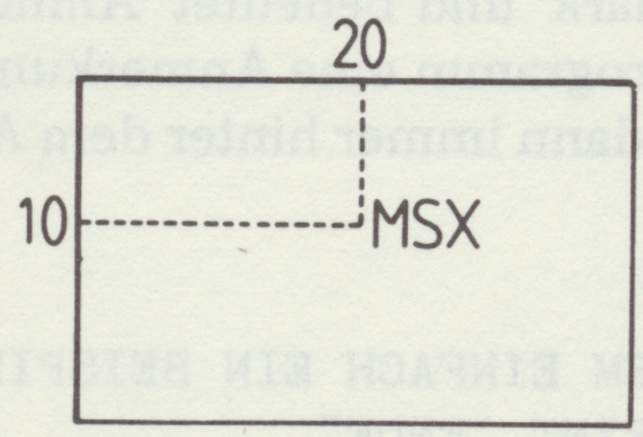
Die Bildschirmaufteilung ist dabei wie folgt:



Beispiel eines Programmes:

```
10 CLS
20 LOCATE 20,10
30 PRINT "MSX"
```

ergibt als Resultat:



Dieses Programm beginnt mit einer noch nicht behandelten Anweisung, der CLS-Anweisung (Clear-Screen-Anweisung). Sie führt zu dem Ergebnis, daß der Schirm gelöscht und der Cursor in die Ecke oben links gesetzt wird. Die anschließende LOCATE-Anweisung versetzt den Cursor in die Position (20,10). Dies bedeutet, daß der Text MSX von dieser Position aus abgebildet wird.

PRINT USING In sehr vielen Bereichen wollen wir, daß die Ergebnisse immer in einer bestimmten Form erscheinen. Man denke beispielsweise an finanzielle Berechnungen. Die Ergebnisse müssen hier immer in einer Form mit zwei Stellen hinter dem Komma gebracht werden.
Die Art und Weise der Abbildung einer Zahl geben wir mit einer sogenannten PRINT USING-Anweisung an.

Sie hat immer den folgenden Aufbau:

```
PRINT USING "Information über Darstellung"; Zahl
```

Wir geben ein Beispiel:

```
PRINT USING "##.##"; 32.7
```

Auf dem Bildschirm erscheint dann

```
32.70
```

Der Ausdruck zwischen den Anführungsstrichen, d.h. ##.##, gibt an, wie die Zahl dargestellt werden muß. In diesem Fall mit zwei Ziffern vor und zwei hinter dem Dezimalpunkt.

```
##.##
```

zwei Ziffern vor dem Punkt — *Punkt* — *zwei Ziffern hinter dem Punkt*

Die Möglichkeiten von PRINT USING sind außerordentlich groß. Eine vollständige Übersicht finden Sie im Anhang D.

REM

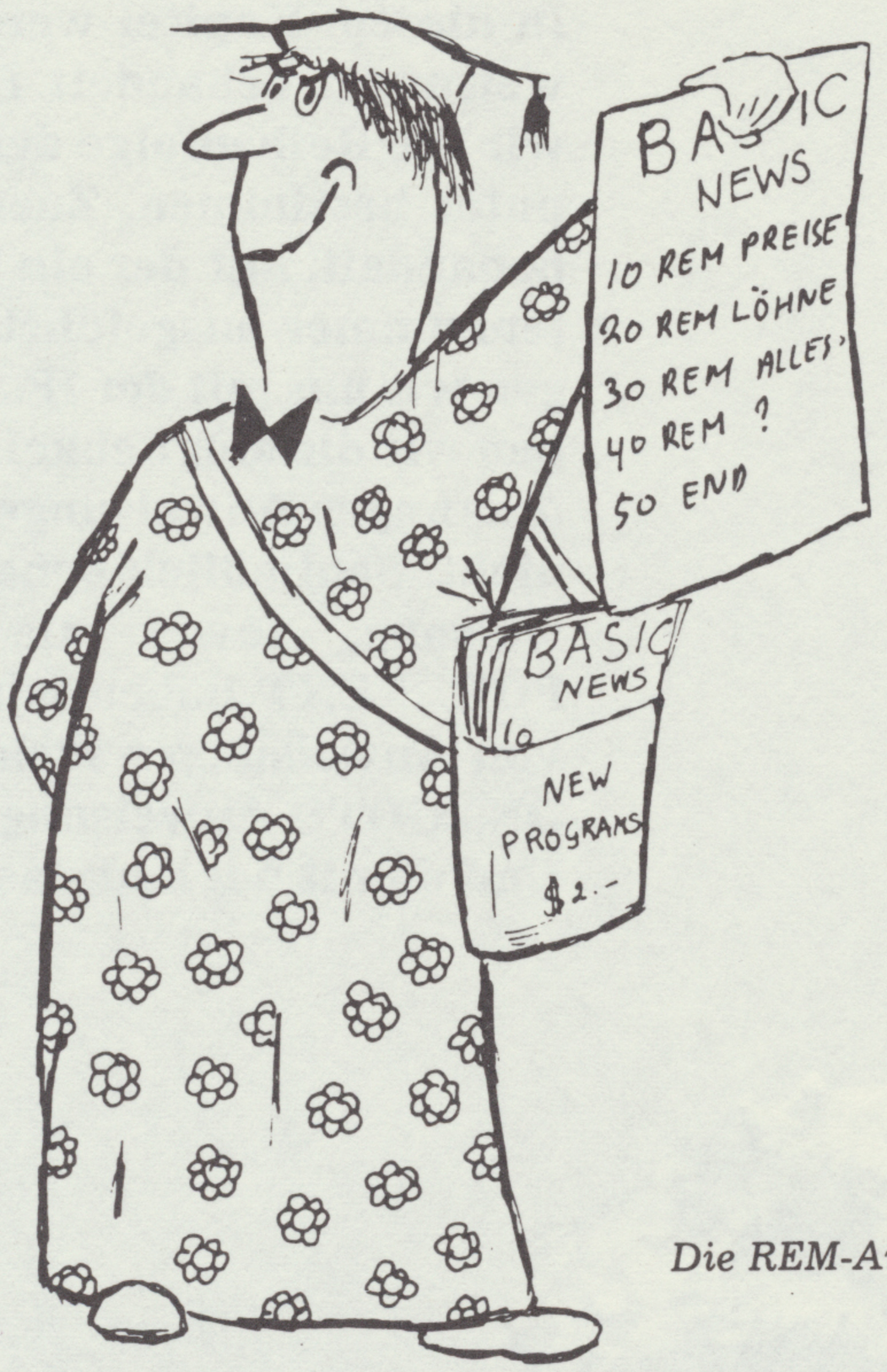
Die letzte Anweisung, die wir in diesem Kapitel besprechen, ist die sogenannte REM-Anweisung. Der Ausdruck REM kommt von 'remark' und bedeutet 'Anmerkung'. Wir können hiermit einem Programm eine Anmerkung zufügen. Dieser Kommentar muß dann immer hinter dem Ausdruck REM gegeben werden.
Beispiel:

```
10 REM EINFACH EIN BEISPIEL
20 PRINT "ENDE"
```

Resultat nach der REM-Anweisung:

```
ENDE
```

Wie wir sehen, wird der hinter REM getippte Kommentar nicht abgedruckt. Wir sehen ihn nur wieder, wenn wir wieder ein LIST-Kommando geben.
Ein Kommentar kann bei längeren Programmen sehr wichtig sein. Wir machen von dieser Möglichkeit Gebrauch, um Sinn und Zweck eines Programmes darzulegen.



Die REM-Anweisung ...

In diesem Kapitel werden die sogenannten Steueranweisungen behandelt. Dies sind Anweisungen, in denen wir die Reihenfolge der Verarbeitung durch den Computer bestimmen. Zuerst wird die GOTO-Anweisung behandelt, mit der ein Sprung zu einer genannten Zeilennummer ausgeführt wird. Anschließend beschäftigen wir uns mit der IF...THEN-Anweisung. Hiermit haben wir die Möglichkeit, aufgrund von bestimmten Bedingungen Anweisungen ausführen zu lassen oder auch nicht. Schließlich behandeln wir die FOR...NEXT-Anweisung, sowie die ON...GOTO-Anweisung. Mit FOR...NEXT haben wir die Möglichkeit, eine Gruppe von Anweisungen mehrmals ausführen zu lassen. Die ON...GOTO-Anweisung erinnert an einen Wahlschalter. Und damit beginnt das wirkliche Programmieren.

10. Die Steueranweisungen: GOTO, IF...THEN, FOR...NEXT und ON...GOTO

Einleitung Bei den bisher behandelten Programmen werden die Anweisungen nacheinander ausgeführt. Diese Anlage war, wie wir sahen, vor allem zur Berechnung von Ergebnissen von Formeln geeignet. In den ersten Anweisungen erhalten bestimmte Variablen der Formel einen bestimmten Wert, und in den letzten Anweisungen wird der Wert der Formel berechnet.
In sehr vielen Fällen wollen wir jedoch die Möglichkeit haben, in die Reihenfolge der Ausführung dieser Anweisungen einzugreifen.
Die Anweisungen mit diesen Möglichkeiten bezeichnen wir als Steueranweisungen, weil sie den Lauf der Programmverarbeitung steuern.

Diese Anweisungen sind:

`GOTO`	zur Ausführung eines Sprunges im Programm
`IF...THEN`	zur Ausführung oder Unterlassung eines Teilprogrammes aufgrund selbst anzugebender Bedingungen.
`FOR...NEXT`	wiederholte Ausführung einer Reihe von Anweisungen
`ON...GOTO`	zur Ausführung eines Sprunges innerhalb einer Reihe möglicher Sprünge.

Diese Anweisungen werden wir im Laufe dieses Kapitels behandeln. Im Kapitel 14 besprechen wir außerdem noch:

`GOSUB...RETURN`	um eine bestimmte Reihe von Anweisungen aus einem beliebigen Punkt des Programmes aufzurufen.

ON...GOSUB	zum Aufrufen einer bestimmten Reihe von Anweisungen aus einer Anzahl möglicher Anweisungsreihen.
DEF FN	zur Definierung einer Funktion.

Die jetzt zu besprechenden Anweisungen wollen wir so einfach wie möglich behandeln. So läßt sich der Effekt solch einer Anweisung am besten übersehen. In den folgenden Kapiteln wird die Anwendung dieser Anweisungen auf Grund realistischer Beispiele erläutert.

GOTO

Die GOTO-Anweisungen wird von allen BASIC-Anweisungen wohl am meisten kritisiert. Die Kritik richtet sich vor allem darauf, daß die GOTO-Anweisung für viele Konstruktionen notwendig ist und demzufolge die Programme unübersichtlich werden. Doch lassen wir diese düsteren Klänge vorerst einmal außeracht, und befassen wir uns mit der Wirkung der Anweisung selbst.
Die GOTO-Anweisung hat eine sehr einfache Form:

GOTO Zeilennummer

Wenn der Computer bei dieser Anweisung anlangt, macht er einen Sprung zur genannten Zeilennummer, und dort wird das Programm fortgesetzt. Da dieser Sprung immer ausgeführt wird, sprechen wir von einem bedingungslosen Sprung.
Ein einfaches Beispiel:

```
10 PRINT "DIES WIRD ABGEDRUCKT"
20 GOTO 40
30 PRINT "DIES NICHT"
40 PRINT "DIES WIRD AUCH ABGEDRUCKT"
```

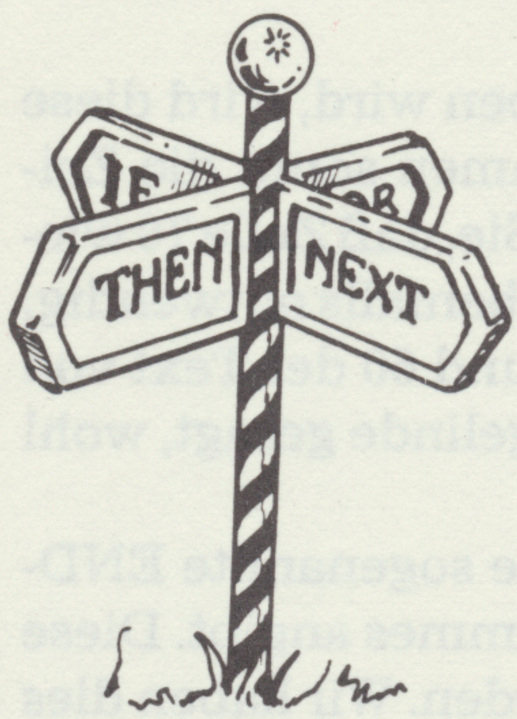
Resultat:

```
DIES WIRD ABGEDRUCKT
DIES WIRD AUCH ABGEDRUCKT
```

Zeile 20 gibt einen Sprung an nach Zeile 40, so daß Zeile 30 immer ausgelassen wird. Diese Anweisung kommt uns im Augenblick vielleicht recht unsinnig vor. Weshalb wollen wir schließlich Anweisungen überspringen, wenn diese offenbar doch nie ausgeführt werden müssen?
Bei der Behandlung der folgenden Anweisung werden wir jedoch erkennen, daß die GOTO-Anweisung in einigen Fällen doch angewandt werden muß.

IF...THEN Die einfachste Form der IF...THEN-Anweisung ist:

IF Bedingung THEN Anweisung



Wie wir sehen, ist zwischen IF und THEN offenbar eine Bedingung angegeben, und WENN (IF) diese Bedingung gültig ist, dann wird die nach DANN (THEN) genannte Anweisung ausgeführt. Wir können dies mit einem sehr einfachen Beispiel verdeutlichen:

```
10 INPUT "ZAHL EINGEBEN";K
20 IF K=3 THEN PRINT "ZAHL WAR DREI"
30 PRINT "ENDE PROGRAMM"
```

Das Programm beginnt mit der INPUT-Anweisung, die besagt, daß eine Zahl eingegeben werden soll. Gehen wir davon aus, daß die Zahl 3 eingegeben wird; K wird also der Wert 3 zugemessen. In der folgenden Zeile steht eine IF...THEN-Anweisung. Die Bedingung lautet:

K=3

oder in Worten ausgedrückt:

ist K gleich 3?

Bitte beachten Sie, daß wir hier die Frageform gebraucht haben. Solch ein Ausdruck kann also nur richtig oder falsch sein. Nehmen wir also unser Beispiel: Ist K tatsächlich gleich 3 oder ist dies nicht der Fall?
Wir sprechen also davon, ob es stimmt oder nicht. Um es noch deutlicher zu sagen: Richtig oder falsch? Nun gut, K war der Wert 3 zugemessen, und so ist die Schlußfolgerung, daß die Bedingung erfüllt worden ist.
In diesem Fall wird der Computer nach THEN die Anweisung ausführen, und als Resultat folgt die Mitteilung 'Zahl war drei'. Wenn K nicht gleich 3 sein würde oder, anders gesagt, die Bedingung nicht erfüllt würde, wird die Anweisung nach THEN einfach übersprungen. Hat der Computer die IF...THEN-Anweisung verarbeitet, führt er einfach die nächste Anweisung aus.

Im folgenden Beispiel zeigen wir ein Programm, in dem auf THEN eine GOTO-Anweisung folgt.

```
10 PRINT "WIEVIEL IST 2+5?"
20 INPUT K
30 IF K=7 THEN GOTO 80
40 PRINT "ABER NEIN"
50 PRINT "DIE ANTWORT AUF 2+5"
60 PRINT "IST NATÜRLICH 7"
70 GOTO 90
80 PRINT "DAS IST DIE RICHTIGE ANTWORT"
90 END
```

Hier sehen wir, daß hinter THEN die Anweisung GOTO 80 steht. Wenn also für K der Wert 7 eingesetzt wird, erfolgt ein Sprung zur Zeile 80.
Wenn für K ein anderer Wert als 7 eingegeben wird, wird diese GOTO-Anweisung übersprungen und kommen somit die Zeilen 40, 50, 60 und 70 an die Reihe. Beachten Sie, daß Zeile 70 wieder eine GOTO-Anweisung enthält. Die ist ebenfalls notwendig, weil sonst nach dem Text von Zeile 40, 50 und 60 der Text von Zeile 80 erscheinen würde - und das wäre, gelinde gesagt, wohl etwas merkwürdig.
Die letzte Anweisung dieses Kapitels ist die sogenannte END-Anweisung, die die Beendigung des Programmes angibt. Diese Anweisung darf eventuell weggelassen werden. Wir haben dies in der Praxis auch bei vorigen Beispielen schon immer getan.

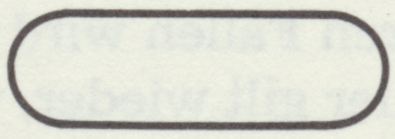
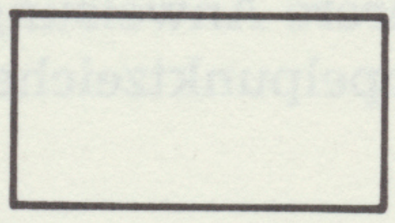
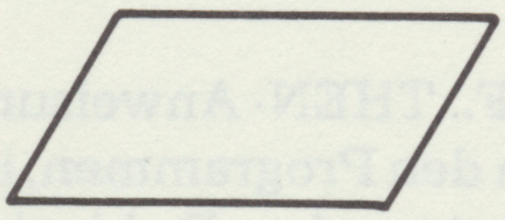
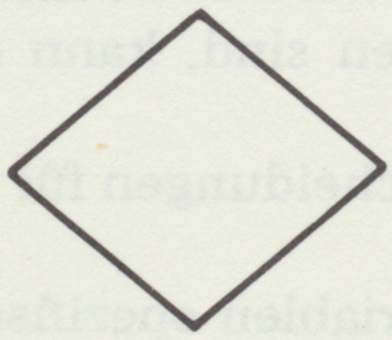
Zu diesem Programm noch folgende Bemerkungen:

- Statt
  IF K=7 THEN GOTO 80
  hätte man auch schreiben können:
  IF K=7 THEN 80
  oder unter Auslassung von THEN
  IF K=7 GOTO 80
- Nach THEN hätten wir eventuell mehrere Anweisungen geben können, jedoch durch Zeichen : getrennt. Dann gilt jedoch die Voraussetzung, daß der ganze IF...THEN- Ausdruck in eine BASIC-Zeile paßt (höchstens 255 Zeichen).
- In der IF...THEN-Anweisung wird K mit 7 verglichen und zwar durch das Zeichen =. Ein derartiges Zeichen in einer Bedingung nennen wir Vergleichsoperator. Außer dem Zeichen = können wir auch folgende Zeichen verwenden:

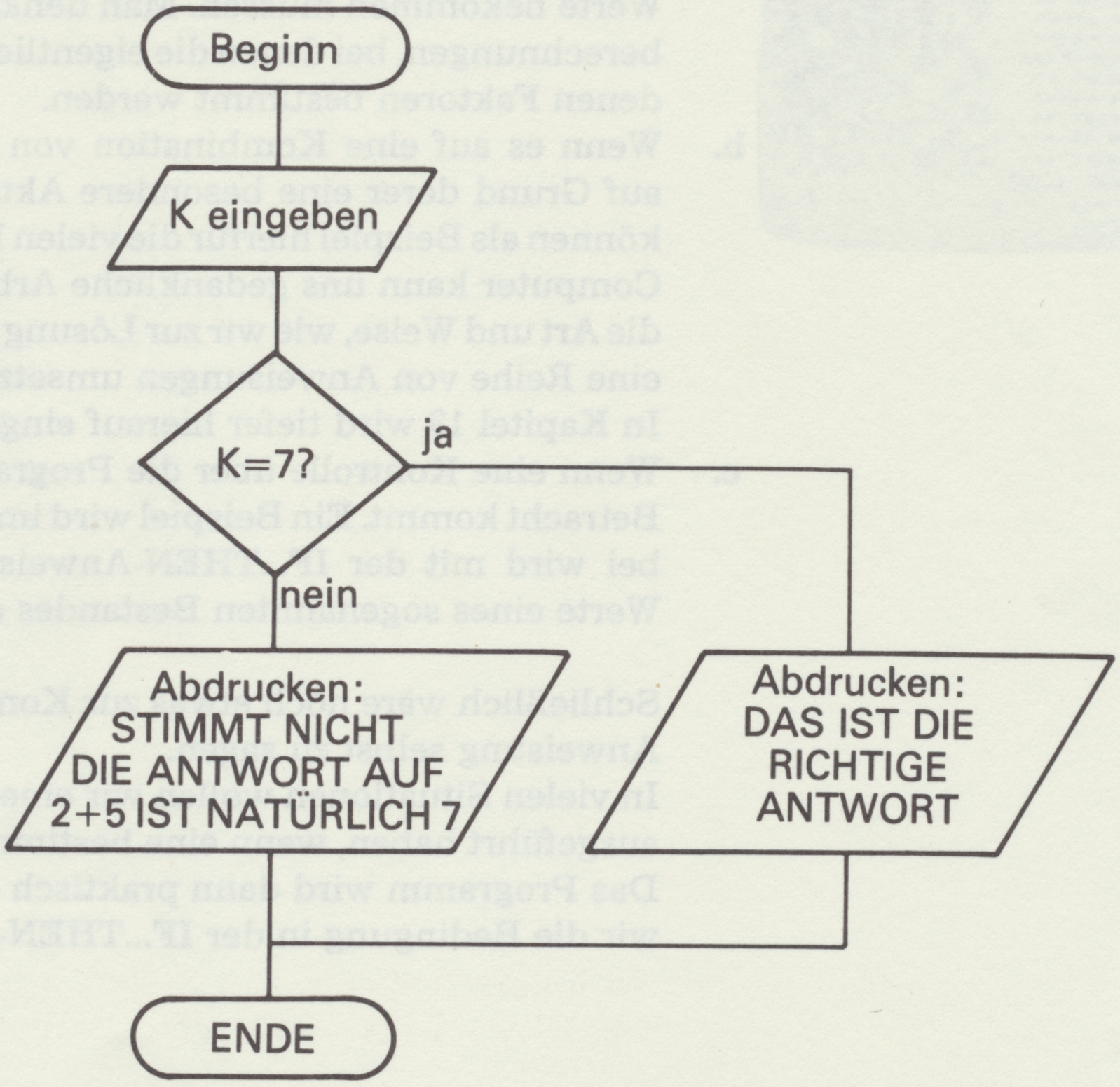
Zeichen	Bedeutung
<	kleiner als
>	größer als
< =	kleiner als oder gleich
= <	wie voriges Zeichen
> =	größer als oder gleich
= >	wie voriges Zeichen
<>	ungleich wie
><	wie voriges Zeichen

- Wir können logische Ausdrücke auch kombinieren.
  Die Möglichkeiten dazu werden im Anhang F besprochen.

Flußdiagramm Zur besseren Darstellung eines Programmverlaufs verwendet man mitunter Flußdiagramme. In der Praxis entwirft man oft zuerst solch ein Flußdiagramm, ehe man dies in ein Programm übersetzt oder, wie man es mitunter ausdrückt, 'in ein Programm kodiert'.
In Flußdiagrammen werden folgende Zeichen verwendet:

Zeichen	*Bedeutung*
	Symbol zur Kennzeichnung von Beginn und Ende eines Programmes
	Symbol einer Bearbeitung wie etwa A=3
	Symbol für Eingabe- oder Ausführungsanweisung (PRINT/INPUT)
	Symbol zur Kennzeichnung einer IF...THEN- Anweisung

Mit Hilfe dieser Symbole können wir jetzt unser letztes Programm folgendermaßen darstellen:



IF...THEN ...ELSE

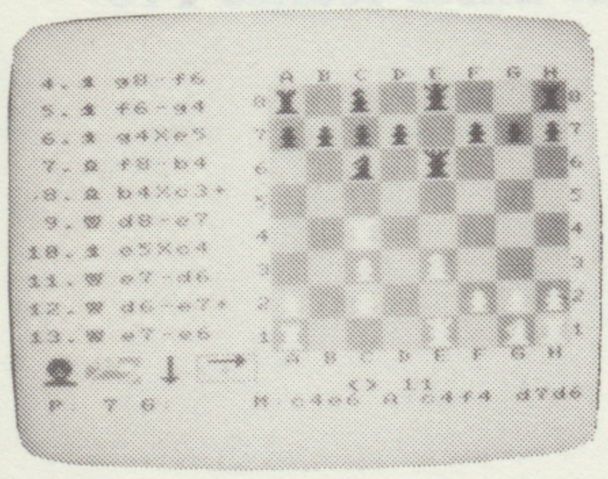
MSX-BASIC erlaubt auch die IF...THEN-Anweisung mit der Erweiterung durch den Ausdruck ELSE. Diese Erweiterung erläutern wir wieder mit einem einfachen Beispiel:

```
10 INPUT K
20 IF K=7 THEN PRINT "K IST 7" ELSE
               PRINT "K IST UNGLEICH 7"
30 END
```

Die Anweisung hinter THEN wird nur ausgeführt, wenn die angegebene Bedingung zutrifft; in allen anderen Fällen wird die Anweisung nach ELSE ausgeführt. Auch hier gilt wieder, wie auch bei THEN, daß wir hinter ELSE mehrere Anweisungen geben können, die jeweils durch das Doppelpunktzeichen : voneinander getrennt werden.

Nachbetrachtung

Der Charakter der Programme, in denen IF...THEN- Anweisungen enthalten sind, unterscheidet sich von den Programmen, in denen Formeln auf direktem Weg berechnet werden. Da hier jedoch Wahlmöglichkeiten eingeführt worden sind, kann das Programm logische Schlüsse ziehen.
Im allgemeinen kann man solche Wahlentscheidungen für nötig halten:



a. wenn, je nach bestimmten Resultaten, Variablen spezifische Werte bekommen müssen. Man denke beispielsweise an Zinsberechnungen, bei denen die eigentlichen Zinsen von verschiedenen Faktoren bestimmt werden.

b. Wenn es auf eine Kombination von Bedingungen ankommt, auf Grund derer eine besondere Aktion stattfinden muß. Wir können als Beispiel hierfür die vielen Denkspiele anführen. Der Computer kann uns gedankliche Arbeit abnehmen, wenn wir die Art und Weise, wie wir zur Lösung kommen können, auch in eine Reihe von Anweisungen umsetzen können.
In Kapitel 13 wird tiefer hierauf eingegangen.

c. Wenn eine Kontrolle über die Programmausführung selbst in Betracht kommt. Ein Beispiel wird im Anhang A gegeben. Hierbei wird mit der IF...THEN-Anweisung kontrolliert, ob alle Werte eines sogenannten Bestandes eingelesen worden sind.

Schließlich wäre noch etwas zur Konstruktion der IF...THEN-Anweisung selbst zu sagen.
In vielen Situationen wollen wir eine Reihe von Anweisungen ausgeführt haben, wenn eine bestimmte Bedingung erfüllt ist. Das Programm wird dann praktisch einfacher gestaltet, wenn wir die Bedingung in der IF...THEN-Anweisung umdrehen.

So ist:

```
IF K=5 Zeilennr. a
....
....
....
GOTO b
Zeilennr. a  Reihe auszuführender
             Anweisungen wie K=5
Zeilennr. b  Anschluß des Programmes
```

das gleiche wie:

```
IF K<>5 THEN Zeilennr. b
             Reihe auszuführender
             Anweisungen
Zeilennr. b  Fortsetzung des Programmes
```

Bei letztgenanntem Beispiel ist die Bedingung K=5 'umgedreht' in K<>5. Wir sehen nun, daß die Reihe der auszuführenden Anweisungen direkt unter der IF...THEN-Anweisung selbst zu stehen kommt, und außerdem wird eine GOTO-Anweisung vermieden.

Das Denken in IF...THEN- und GOTO-Anweisungen kann mitunter verwirrend wirken. Häufig ergibt sich eine Situation, die sich in Worten folgendermaßen ausdrücken ließe:
SOLANGE die Bedingung zutrifft, Anweisungsreihe AUSFÜHREN.
Es ist sinnvoll, diese Konstruktion bei der Aufstellung eines Programmes einzuhalten und erst in letzter Instanz die Umsetzung in IF...THEN- und GOTO-Anweisungen vorzunehmen.
Ein Beispiel:
Gehen wir einmal davon aus, daß eine Reihe einzuführender Zahlen zusammengezählt werden soll, solange die einzuführenden Zahlen positiv sind. Das 'Programm' würde dann lauten:

```
INPUT K
SOLANGE K>0 AUSFÜHREN
            SUMME=SUMME+K
            INPUT K
```

In BASIC übersetzt wird dieses Programm:

```
10 INPUT K
20 IF K <=0 THEN STOP
30 SUMME=SUMME+K
40 GOTO 10
```

(Anmerkung: die Anweisung STOP hat Abbruch des Programmes zur Folge.)
Hierzu ist anzumerken, daß es bei diesem Beispiel sinnvoll ist, den Wert von SUMME zu Beginn des Programmes auf 0 (Null) zu setzen (Zuweisung eines Initialwertes), beispielsweise:

```
5 SUMME=0
```

Das Arbeiten mit der SOLANGE...AUSFÜHREN-Konstruktion kann bei der Aufstellung von Programmen wesentliche Vorteile haben, weil die Struktur hierdurch viel deutlicher wird. Durch die Kombination von IF...THEN- und GOTO- Anweisungen wird ein Programm rasch unübersichtlich.

FOR...NEXT

Die FOR...NEXT-Anweisung können wir als ein besonders praktisches Programmierwerkzeug ansehen. Hiermit geben wir an, daß eine bestimmte Serie von Anweisungen mehrmals wiederholt werden soll.
Folgendes Beispiel macht dies deutlich:

```
10 FOR A=1 TO 5
20 PRINT A; "DIES IST EINE DEMONSTRATION"
30 NEXT A
```

Resultat:

```
1 DIES IST EINE DEMONSTRATION
2 DIES IST EINE DEMONSTRATION
3 DIES IST EINE DEMONSTRATION
4 DIES IST EINE DEMONSTRATION
5 DIES IST EINE DEMONSTRATION
```

Die Anweisungen, die wiederholt ausgeführt werden müssen, werden immer eingeschlossen zwischen der Zeile, die mit FOR beginnt, und der mit NEXT endenden Zeile, also schematisch:

```
.. FOR Name Variable=...
.. ...
.. ...  } Diese Anweisungen wiederholt ausführen
.. ...
.. NEXT Name der Variablen
```

In unserem Fall handelt es sich nur um eine Anweisung, nämlich um die von Zeile 20. Wie oft diese Anweisungen ausgeführt werden sollen, kann man völlig durch das bestimmen, was in der mit FOR beginnenden Zeile beschrieben ist.

Bei

```
FOR A=1 TO 5
```

wird der Teil fünfmal wiederholt, wobei A nacheinander den Wert 1, 2, 3, 4, 5 annimmt. Die hinter FOR angegebene Variable bezeichnen wir daher auch zweckmäßig als 'Zähl-Variable'.
In diesem Fall wird A jeweils um 1 erhöht. Wir können jedoch auch eine Schrittgröße angeben, indem wir die Zeile mit FOR mit dem Ausdruck STEP ergänzen:

```
10 FOR A=1 TO 6 STEP 2
20 PRINT A; "DIES IST EINE DEMONSTRATION"
30 NEXT A
40 PRINT A
```

Resultat:

```
1 DIES IST EINE DEMONSTRATION
3 DIES IST EINE DEMONSTRATION
5 DIES IST EINE DEMONSTRATION
7
```

Hier bekommt also die Zähl-Variable A nacheinander den Wert 1, 3 und 5 und wird 'die Schleife' für diesen Wert stets weitergeführt.
Wenn A gleich 7 ist, wird die in Zeile 10 gesetzte Grenze (6) überschritten und setzt der Computer das Programm mit Zeile 40 fort. Wir lassen nochmals den Wert von A und damit den Wert 7 abdrucken.
Die aus diesem Programm zu ziehende Lehre ist klar: aufpassen, wenn wir nach einer FOR...NEXT-Anweisung die Zähl-Variable weiter im Programm anwenden; sie hat nicht unbedingt den Endwert, der in der Zeile mit FOR angegeben wird!

Noch einige Beispiele:

- `FOR A=10 TO 5 STEP-1`
  Die Anweisungsreihe wird nun ausgeführt mit
  A=10,9,8,7,6 und 5
- `FOR A=-15 TO 15 STEP 3`
  Die Anweisungsreihe wird nun ausgeführt mit
  A=-15,-12,-9,-6,-3,0,3,6,9,12 und 15
- `FOR A=1.4 TO 1.7 STEP .05`
  Die Anweisungsreihe wird nun ausgeführt mit
  A=1.4,1.45,1.50,1.55,1.60,1.65 und 1.70
- `FOR A=B TO C STEP K/L`

Wir sehen hier, wie der Anfangs- und der Grenzwert als Schrittgröße durch Variablen angegeben wird. Die Schrittgröße wird selbst durch eine mathematische Bezeichnung (Teilung von 2 Variablen) angegeben. Es ist durchaus erlaubt, Anfangs-, Grenz- und Schrittwert durch derartige Bezeichnungen anzugeben.

- FOR...NEXT-Anweisungen dürfen selbst auch wiederum FOR...NEXT-Anweisungen enthalten. Voraussetzung ist jedoch, daß die eine Anweisung die andere vollständig umfaßt. So ist die Konstruktion

```
FOR K=1 TO 10
...
FOR J=1 TO 5
...
...
NEXT J
NEXT K
```

Innenschleife wird ganz von Außenschleife umschlossen

durchaus statthaft, dagegen aber ist die Konstruktion

```
FOR K=1 TO 10
...
FOR J=1 TO 5
...
...
NEXT K
...
NEXT J
```

'K-Schleife'

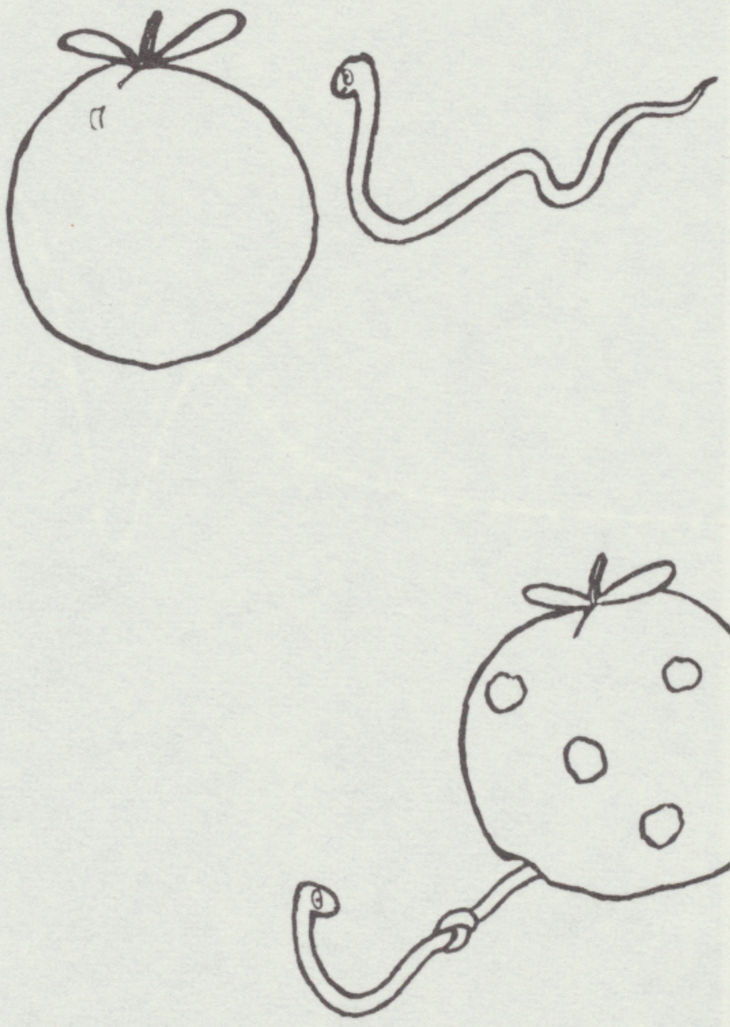
'J-Schleife'

verboten. Hier wird die Innenschleife nicht von der Außenschleife umschlossen.

ON...GOTO Die letzte Anweisung, die wir in diesem Kapitel besprechen, ist die ON...GOTO-Anweisung. Hierbei können wir an einen Wahlschalter denken.
Aufgrund des zwischen ON und GOTO stehenden Wertes wird ein bestimmter Sprung ausgeführt.
Ein Beispiel macht dies deutlich:

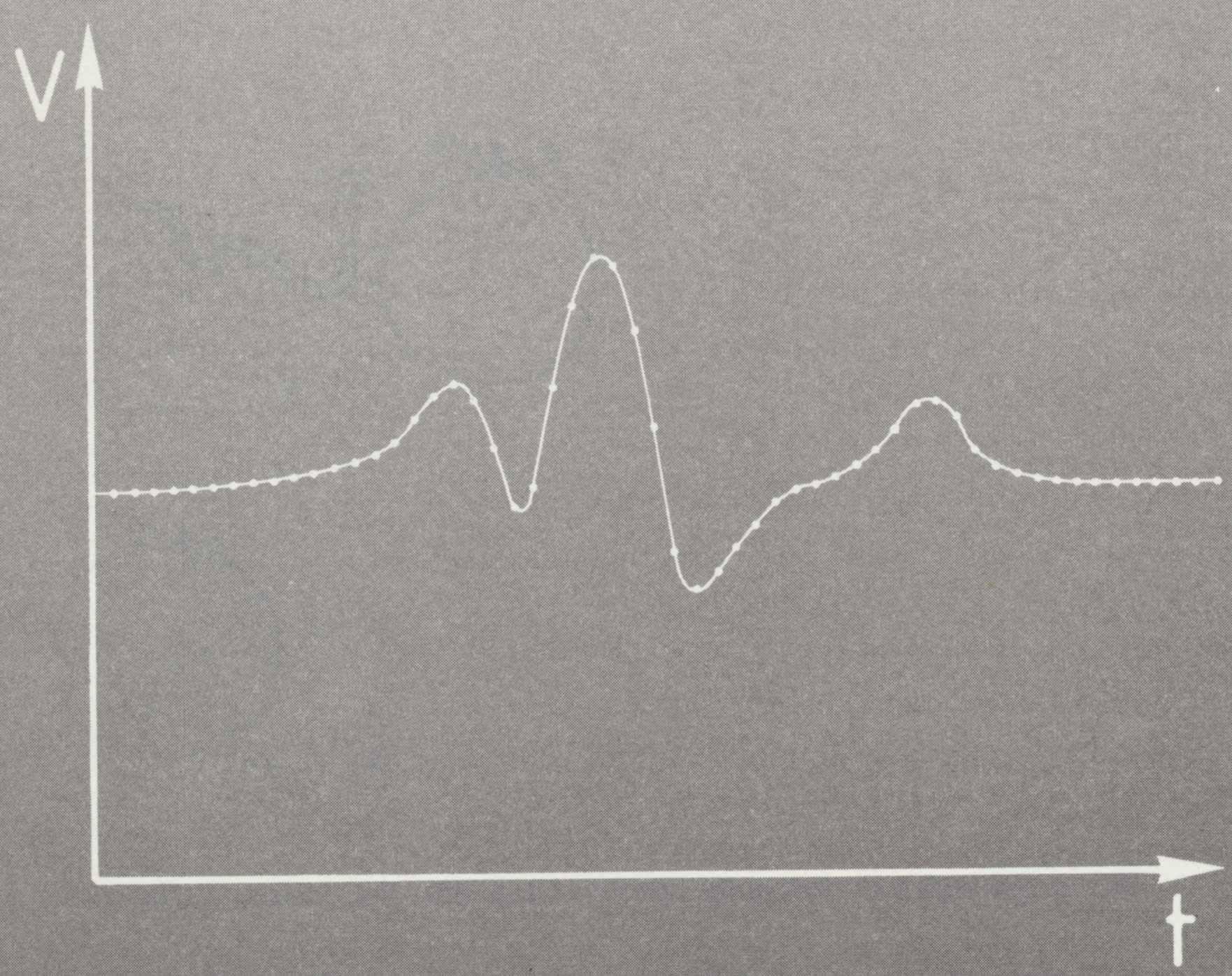
```
10 INPUT K
20 ON K GOTO 30,50
30 PRINT "K IST 1"
40 GOTO 70
50 PRINT "K IST 2"
60 GOTO 70
70 END
```

Wenn K gleich 1 ist, erfolgt ein Sprung zur ersten angegebenen Zeilennummer. Ist K gleich 2, erfolgt ein Sprung zur zweiten Zeilennummer
In diesem Beispiel werden nach GOTO (Zeile 20) nur 2 Zeilennummern genannt, doch in der Praxis können dies viel mehr sein. Zu beachten ist ebenfalls, daß die verschiedenen Programmteile, zu denen gesprungen werden soll, wieder abgeschlossen werden von einer GOTO- Anweisung. Die GOTO-Anweisungen sind hier wieder unvermeidlich.



Wenn Schleifen vorkommen können, muß man besonders aufpassen.

Das Array erschließt die Möglichkeit, auf einfache Weise eine große Vielfalt von Variablen in ein Programm einzuführen. Arrays werden sowohl in administrativen als auch mathematischen Bereichen häufig angewandt. Die Kombination der FOR...NEXT-Anweisung mit einem Array ergibt sich häufig in der Praxis. Am Ende dieses Kapitels wird das durch ein klassisches Problem illustriert: das Suchen von Primzahlen.



11. Array (Auflistung)

Einleitung

In verschiedenen Fällen besteht das Bedürfnis, in einem Programm über eine Vielzahl von Variablen verfügen zu können. Man denke in diesem Zusammenhang beispielsweise an Vorratsverwaltung. Wenn für jeden Artikel, der verwaltungsmäßig erfaßt wird, eine gesonderte Reihe von Variablen eingeführt werden müßte, würde das Programm sehr umfangreich.
Wir besprechen in der Folge, wie BASIC diese Probleme durch die Möglichkeit ausgleicht, über nur eine Anweisung eine Vielzahl von Variablen einzusetzen.

DIM-Anweisung

Die DIM-Anweisung bietet die Möglichkeit zur einfachen Anwendung zahlreicher Variablen. Die Form ist immer so:

```
DIM     Name (Anzahl)
```

beispielsweise

```
DIM A(100)
```

Mit obiger Anweisung führt BASIC eine Reihe von 101 Variablen ein. Der Name jeder dieser Variablen ist dann zusammengesetzt aus dem in der DIM-Anweisung gegebenen Namen und einer eingeklammerten Zahl. So sind

```
A(0) A(1) A(2) A(3) ... A(99) A(100)
```

genau die 101 Variablen, die zur Anweisung DIM A(100) gehören. Wir sprechen in diesem Fall auch vom Array A (Auflistung A), das in diesem Fall aus den Elementen (A0) bis A(100) zusammengesetzt ist.
Alle diese Variablen können wir auf gleiche Weise wie unsere 'normalen' Variablen anwenden.

Beispiel einer EKG-Aufnahme, d.h. also der Aufzeichnung elektrischer Aktivitäten unseres Herzmuskels. Zur Computerverarbeitung derartiger Kurven werden diese erst in eine Reihe von Punkten umgesetzt.

Ein Beispiel:

```
10 DIM A(100)
20 A(3)=6
30 A(27)=5
40 A(98)=A(3)+A(27)
50 PRINT A(98)
```

Resultat:

```
11
```

Dieses merkwürdig anmutende Programm zeigt, daß wir die Auflistungselemente A(3), A(27) und A(98) eingesetzt haben, als handele es sich um 'normale' Variablen mit beispielsweise den Namen A, B und C.

Die Möglichkeiten der Auflistungsvariablen werden vor allem dadurch bestimmt, daß wir die eingeklammerten Nummern (oft Indexzahlen genannt) auch indirekt angeben dürfen.
Beispiele:

`A(93)`	Die Nummer wird hier direkt durch eine Zahl angegeben.
`A(K)`	Die Nummer wird indirekt durch eine Variable angegeben.
`A(K+3)`	Die Nummer wird hier indirekt durch einen Ausdruck angegeben.

Das folgende Beispiel gibt eine einfache Illustration:

```
10 DIM B(20)
20 FOR K=1 TO 20
30 B(K)=K
40 NEXT K
50 FOR K=20 TO 1 STEP -1
60 PRINT B(K);
70 NEXT K
```

Resultat:

```
20 19 18 17 16 15 14 13 12
11 10 9 8 7 6 5 4 3 2 1
```

In Zeile 10 wird Array B eingeführt. Die Zeilen 20 bis 40 geben an, daß B(1) der Wert 1, B(2) der Wert 2 usw. zugewiesen wird. Die Zeilen 50 bis 70 beweisen, daß diese Werte tatsächlich zugewiesen worden sind, denn demzufolge werden die Werte der Variablen B(20), B(19), B(18) usw. gedruckt. Wir bemerken zugleich, wie praktisch die Anweisung FOR...NEXT in Verbindung mit Arrays verwendet werden kann.

Zum Array wollen wir noch folgende Anmerkung machen:

- Wenn in einem Programm eine Feldvariable, z.B. A(6) angewandt wird, ohne daß im Programm eine DIM- Anweisung enthalten ist, nimmt der Computer eine Höchstgrenze von 10 an. Bei dem Beispiel geht der Computer also davon aus, daß die Variable A(6) zur Auflistung A gehört, deren Höchstgrenze 10 ist (DIM A(10).

- Wir dürfen auch sogenannte mehrdimensionale Arrays in einem Programm anwenden, d.h. es werden eingeklammert mehrere Zahlen angegeben. So enthält die Anweisung:

```
DIM A(3,3)
```

  die Variablen:

```
A(0,0)    A(0,1)    A(0,2)    A(0,3)
A(1,0)    A(1,1)    A(1,2)    A(1,3)
A(2,0)    A(2,1)    A(2,2)    A(2,3)
A(3,0)    A(3,1)    A(3,2)    A(3,3)
```

  Bei der Anwendung mehrdimensionaler Arrays stoßen wir meist rasch auf die Begrenzungen des Speichers.

- In einer Zeile dürfen wir auch mehrere Arrays eingeben. Beispiel:

```
10 DIM A(100), P(300), Z(50), P!(30), B%(5)
```

- Den reservierten Platz können wir später frei machen (etwa im Zusammenhang mit Problemen der Speicherkapazität), indem wir die Anweisung ERASE gebrauchen.
  Beispiel:

```
ERASE A,P
```

Berechnung von Primzahlen. Ein Beispiel

Eine Primzahl ist eine Zahl, die nur durch 1 und sich selbst teilbar ist. Aus dieser Definition ergibt sich, daß 1 eine Primzahl ist. Die nächste Zahl 2 ist ebenfalls eine Primzahl, denn sie ist auch nur durch 1 und 2 teilbar. Auch 3 ist eine Primzahl, aber 4 nicht, denn diese Zahl kann man wiederum durch die Primzahl 2 teilen. Die folgende Zahl 5 ist wieder eine Primzahl, während 6 wieder durch 2 und durch 3 teilbar ist und daher nicht zu den Primzahlen gehört.
Überlegen wir uns einmal eine Möglichkeit, wie wir beispielsweise bei den Zahlen 1 bis 250 ermitteln können, welche Zahl eine Primzahl ist. Zuerst schauen wir uns einmal an, wie wir bei den Zahlen 1 bis 10 feststellen können, welche Zahlen Primzahlen sind.

Zu diesem Zweck notieren wir zuerst einmal die Zahlen
1 2 3 4 5 6 7 8 9 10
Nun suchen wir von links nach rechts die Primzahlen auf, und immer, wenn wir solch eine Primzahl finden, streichen wir alle Vielfachen dieser Primzahl durch. Wir wissen, daß 1 eine Primzahl ist, ebenso die folgende Zahl 2. Anschließend streichen wir durch:
1 2 3 ~~4~~ 5 ~~6~~ 7 ~~8~~ 9 ~~10~~
Nun gehen wir zur nächsten Zahl, zur 3. Sie ist noch nicht durchgestrichen, also ist sie eine Primzahl. Wir streichen nun auch alle Vielfachen der 3 durch:
1 2 3 ~~4~~ 5 ~~6~~ 7 ~~8~~ ~~9~~ ~~10~~
Die folgende Zahl ist 4; sie ist durchgestrichen, also keine Primzahl. Die nächste Zahl ist 5, noch nicht durchgestrichen, also eine Primzahl. Die 6 als nächste Zahl ist wieder durchgestrichen usw.
Wir sehen, daß wir mit dem Durchstreichverfahren gewissermaßen alle Primzahlen aus der ursprüngliche Zahlenreihe heraussieben. Schon die alten Griechen kannten übrigens dieses Rezept, denn es ist bekannt als das 'Sieb des Eratosthenes'.

Das nun folgende Programm hat das oben besprochene Rezept zur Grundlage.

```
10 DIM N(250), P(250)
20 FOR J=2 TO 250
30 IF N(J)<0 THEN GOTO 70
40 AP=AP+1
50 P(AP)=J
60 FOR K=J TO 250 STEP J:N(K)=-1:NEXT K
70 NEXT J
80 REM AUSDRUCKEN
90 FOR K=1 TO AP: PRINT P(K);:NEXT K
```

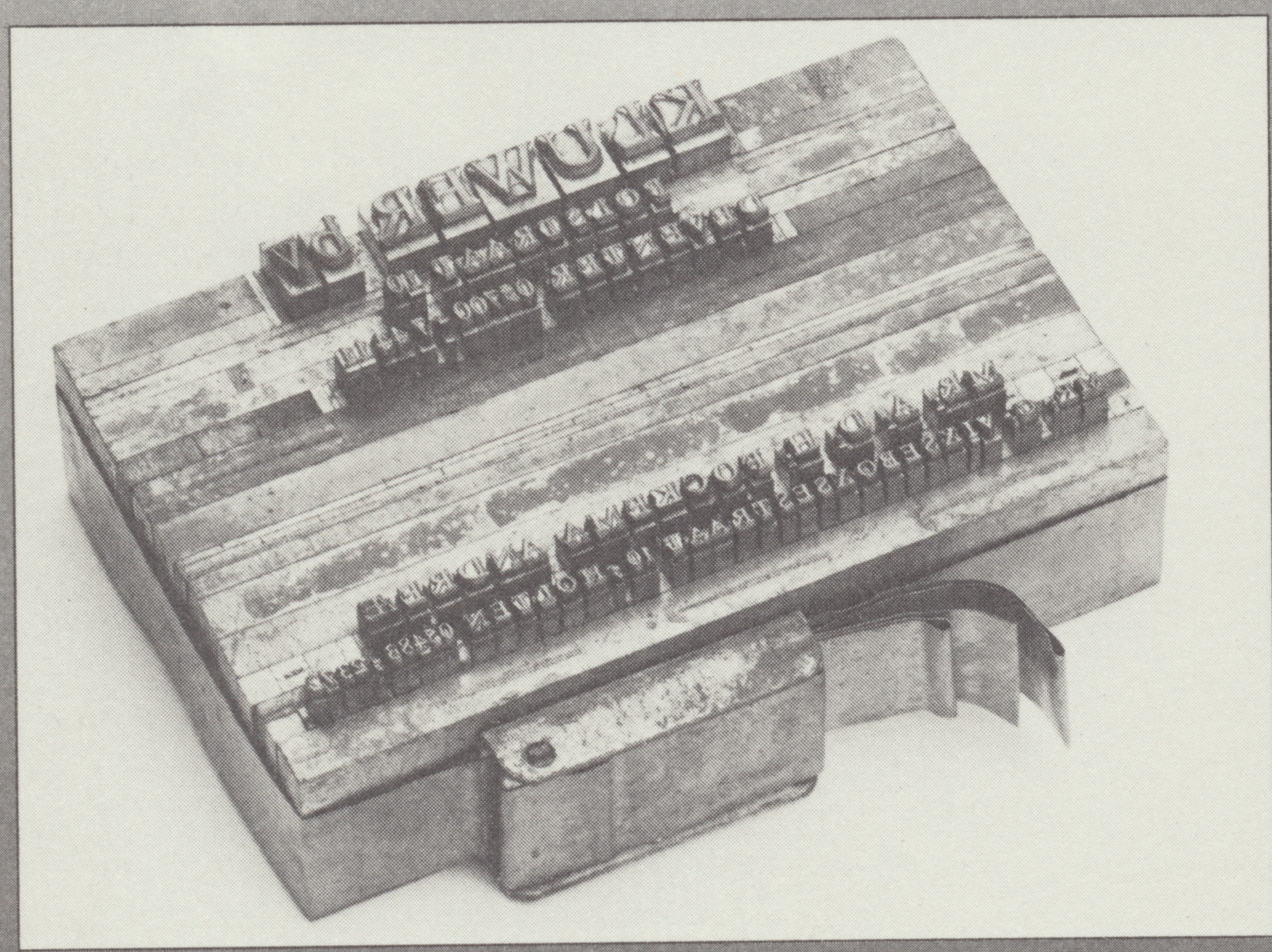
Resultat:

```
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59
61 67 71 73 79 83 89 97 101 103 107 109 113 127
131 137 139 149 151 157 163 167 173 179 181 191
193 197 199 211 223 227 229 233 239 241
```

In Zeile 10 werden zwei Arrays eingebracht. Array N soll behalten, welche Zahlen wir durchgestrichen haben. Wir tun dies in unserem Programm, indem wir solch einer Zahl den Wert –1 zuweisen (Zeile 60). Array P speichert die gefundenen Primzahlen. Dies sind natürlich weniger als 250, doch da wir die genaue Anzahl nicht im voraus wissen, haben wir dieses Array sehr reichlich bemessen.

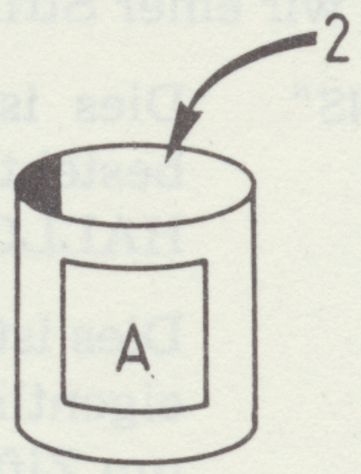
Zeile 20 gibt an: von links nach rechts verschieben. Ist eine Zahl durchgestrichen (Zeile 30), wird die folgende Zahl überprüft. Wenn kein Strich gefunden wird, wird AP um 1 erhöht.
Mit AP zählen wir die Anzahl der Primzahlen. Anschließend wird die gefundene Primzahl im Array P gespeichert (Zeile 50) und werden die Vielfachen durchgestrichen (Zeile 60). Die letzten Zeilen sind hinzugefügt, um die von unserem 'Sieb' aufgefangenen Primzahlen auszudrucken.

In diesem Kapitel gehen wir auf die Möglichkeit ein, den Variablen Buchstabenreihen zuzuweisen. Solch eine Reihe Buchstaben - oder etwas allgemeiner - Zeichen bezeichnen wir als String oder Kette, und die Variablen, denen wir eine derartige Verkettung zuweisen, bezeichnen wir als String-Variablen. Es wird aufgezeigt, daß wir hinsichtlich der Verkettung nur ein Verarbeitungszeichen haben, nämlich +. Daneben gibt es jedoch auch viele Funktionen, die wir bei Strings anwenden können.



12. Strings ... spielen mit Buchstaben

Einleitung Wir haben bisher gesehen, daß wir einer Variablen eine Zahl zuweisen können. Bei der Darlegung des Funktionierens hatten wir das Bild einer Schachtel gebraucht. So wurde die Anweisung LET A=2 folgendermaßen anschaulich gemacht:



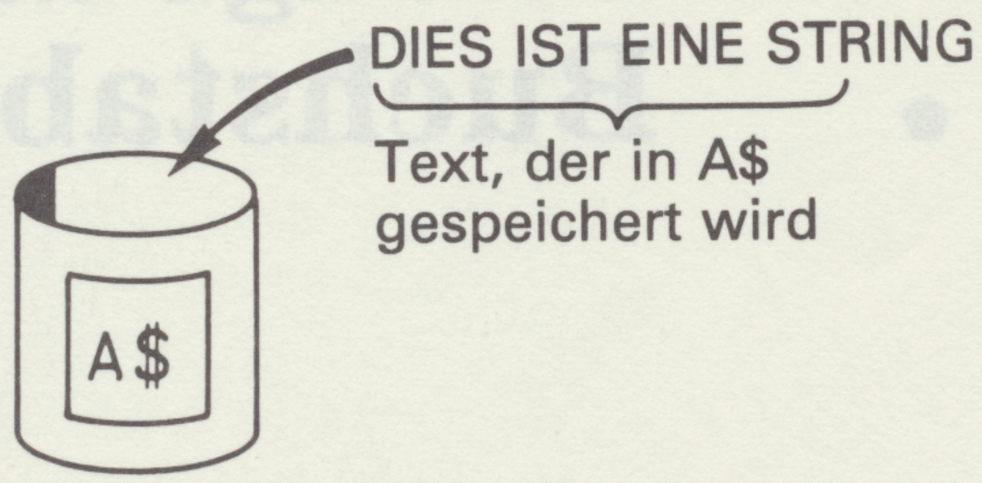
Variablen können wir auch Buchstabenreihen zuweisen. Eine derartige Buchstabenkette bezeichnen wir als String, und dies ist zugleich der Grund, weshalb wir solche Variablen als String-Variable bezeichnen.
Wir müssen natürlich auch durch eine besondere Schreibweise deutlich machen, daß es sich um eine String-Variable handelt. Dafür gilt eine einfache Regel:

Ein wenig romantisch...
Hier sehen wir, wie bei Drucksachen die Buchstaben und dergleichen in gesonderten Blöcken gesetzt werden. Beim Satz spielen Computer eine wichtige Rolle, und damit ist in vielen Druckereien der Setzereikasten (der Kasten mit den Blöcken) verschwunden.

Wenn wir unmittelbar hinter den Namen einer Variablen ein Dollarzeichen setzen, handelt es sich um eine String-Variable. So sind A$, ERSTE$ und TEXT$ Beispiele von Namen, die sich offenbar auf String-Variablen beziehen.
Wir müssen den Text, den wir einer derartigen String-Variablen zuordnen, immer in Anführungszeichen setzen.
Folgendes Programm ist hierfür ein Beispiel:

```
10 LET A$= "DIES IST EINE STRING"
20 PRINT A$
```

Zeile 10 können wir wieder bildlich verdeutlichen:



Achten Sie darauf, daß dieser Text in Zeile 10 auch wirklich zwischen Anführungszeichen gesetzt ist. Betrachten wir jetzt das Resultat des Programmes, so fällt uns auf, daß die Anführungszeichen nicht ausgedruckt werden, sondern das Resultat ist:

```
DIES IST EINE STRING
```

Wir können also schlußfolgern, daß die Anführungsstriche nicht zur eigentlichen String gehören.

Nachstehend noch einige Beispiele von Zeichenfolgen oder Strings, die wir einer String-Variablen zuweisen können:

"HALLO HANS"	Dies ist eine String, die aus 10 Zeichen besteht, denn der Zwischenraum zwischen HALLO und HANS zählt auch mit!
"164"	Dies ist eine String aus drei Zeichen. Es ist eigentlich recht verräterisch, daß es sich hier um Ziffern handelt, denn der Computer erkennt nur Zeichen und keine Ziffern! Wir werden noch sehen, wie wir in solchen Sonderfällen eine derartige 'Scheinzahl' in eine wirkliche Zahl umwandeln können.
" "	Dies ist eine String, die aus keinem einzigen Zeichen besteht, da die Anführungszeichen direkt hintereinander stehen. Wir werden noch an Beispielen lernen, was für Vorteile solch eine 'leere String' bieten kann. Diese String wird gelegentlich auch als Null-String bezeichnet.

Verkettung oder Aneinanderreihen von Strings

Mit dem Pluszeichen + können wir angeben, daß zwei Strings miteinander verbunden werden sollen. Computerleute sprechen dann von Verkettung.

Ein Beispiel macht dies deutlich:

```
10 A$="MSX-"
20 B$="BASIC"
30 C$=A$+B$
40 PRINT C$
```

Resultat:

```
MSX-BASIC
```

Wir sehen also, wie in den Zeilen 10 und 20 die Strings 'MSX-' und 'BASIC' A$ und B$ zugewiesen werden. In der anschließenden Zeile 30 werden diese Strings 'zusammengezählt' und C$ zugewiesen.

Stringfunktionen

Bisher konnten wir noch nicht viel mit Strings anfangen, sondern höchstens zwei oder mehrere Strings miteinander verketten, um auf diese Weise eine längere String zu bekommen. MSX-BASIC hat glücklicherweise eine Vielzahl von Funktionen, mit denen allerlei String-Vorgänge ermöglicht werden. Diese Stringfunktionen können wir in zwei Gruppen unterteilen:

Gruppe 1: Funktionen, die wiederum einen String als Resultat haben. Die Namen dieser Funktionen enden immer mit $.

Gruppe 2: Funktionen, bei denen eine Zahl das Resultat ist. Die meisten Funktionen dieser Gruppe machen Gebrauch von einer bestimmten Regel. Diese Regel bezieht sich auf Zuweisung von Zahlen an Buchstaben oder sonstige Zeichen. Die Namen dieser Funktionen enden nicht mit einem $-Zeichen.

LEN

Diese Funktion endet nicht mit einem $-Zeichen, und offensichtlich handelt es sich hier um eine Funktion, deren Ergebnis eine Zahl ist. So ist es, denn mit der LEN-Funktion bestimmen wir nämlich die Anzahl der Zeichen je String.
Wieder ein Beispiel:

```
10 A$="MSX"
20 B=LEN (A$)
30 PRINT B
```

Resultat:

```
3
```

In Zeile 10 wird A$ der String 'MSX' zugewiesen. Dieser String besteht also aus drei Zeichen. In der nächsten Zeile wird B das Resultat von LEN(A$) zugewiesen, d.h. die Anzahl der Zeichen, aus denen A$ zusammengesetzt ist.
Das Resultat bestätigt dies.

**LEFT$** Die LEFT$-Funktion ist die erste Stringfunktion, mit der wir gewissermaßen aus einer String einen Teil herausschneiden können. Solchen Teil einer String bezeichnen wir als Substring. Folgendes Beispiel illustriert die LEFT$-Funktion:

```
10 A$="MSX-BASIC"
20 B$=LEFT$(A$,3)
30 PRINT B$
```

Resultat:

```
MSX
```

Die LEFT$-Funktion von Zeile 20 gibt an, daß von String A$ die ersten drei Zeichen genommen werden müssen, und das Resultat bestätigt dies.

**RIGHT$** Wieder eine String-Funktion zur Erzielung einer Substring. Während es sich bei der LEFT$-Funktion um die ersten Zeichen einer genannten String handelte, kommt es bei der RIGHT$-Funktion auf die letzten Zeichen an.
Wieder ein Beispiel:

```
10 A$="MSX-BASIC"
20 B$=RIGHT(A$,5)
30 PRINT B$
```

Resultat:

```
BASIC
```

In Zeile 20 wird mit RIGHT$(A$,5) angegeben, daß B$ die letzten fünf Zeichen von A$ zugewiesen werden müssen.
Das folgende Beispiel vermittelt eine weitere Illustration der LEFT$- wie der RIGHT$-Funktion.

```
10 A$="MSX-BASIC"
20 N=LEN(A$)
30 FOR K=1 TO N
40 B$=LEFT$(A$,K)
50 C$=RIGHT$(A$,K)
60 PRINT B$,C$
70 NEXT K
```

Resultat:

```
M            C
MS           IC
MSX          SIC
MSX-         ASIC
MSX-B        BASIC
MSX-BA       -BASIC
MSX-BAS      X-BASIC
MSX-BASI     SX-BASIC
MSX-BASIC    MSX-BASIC
```

**MID$** Während wir mit LEFT$ und RIGHT$ den ersten bzw. letzten Teil einer String 'abschneiden' können, haben wir mit MID$ die Möglichkeit, einen beliebigen Teil aus einer String herauszuschneiden. Hierbei geben wir natürlich zuerst an, um welche String es sich handelt, anschließend wird das Zeichen angegeben, von dem aus die String herausgeschnitten werden soll, und eventuell gibt man dann noch ein drittes Zeichen an: diese Zahl gibt an, aus welcher Zeichenzahl die Substring bestehen soll. Lassen wir letztgenannte Zahl weg, so werden alle Zeichen von der angegebenen Stelle an genommen. Das folgende Beispiel macht dies deutlich:

```
10 A$="MSX-BASIC"
20 B$=MID$(A$,2,3)
30 C$=MID$(A$,2)
40 PRINT B$
50 PRINT C$
```

Resultat:

```
SX-
SX-BASIC
```

Wie wir erkennen, wird in Zeile 20 eine Substring aus drei Zeichen vom zweiten Zeichen an aufgebaut. In Zeile 30 wird in der MID$-Funktion die dritte Zahl weggelassen. Wir sehen, wie nun der 'Rest' vom zweiten Zeichen an als Substring genommen wird.

ASC Diese Funktion endet nicht mit dem $-Zeichen, und hieraus ist zu ersehen, daß das Ergebnis offenbar eine Zahl ist.
Das stimmt: die Zahl, um die es sich hier handelt, ist der sogenannte ASCII-Code. Nach einer bestimmten Regel wird jedem Zeichen (Buchstabe, Ziffer, Satzzeichen) eine Zahl zugewiesen (siehe Anhang B).
In nachstehender Tabelle sind die ASCII-Zahlen der Großbuchstaben A-Z ersichtlich.

ASCII-Werte der Großbuchstaben A-Z

Buchstabe	*Zahl*	*Buchstabe*	*Zahl*	*Buchstabe*	*Zahl*	*Buchstabe*	*Zahl*
A	65	H	72	O	79	V	86
B	66	I	73	P	80	W	87
C	67	J	74	Q	81	X	88
D	68	K	75	R	82	Y	89
E	69	L	76	S	83	Z	90
F	70	M	77	T	84		
G	71	N	78	U	85		

Aus dieser Tabelle ist ersichtlich, daß beispielsweise A der Zahl 65 entspricht und K der Zahl 75. Die Tatsache, daß jedem Buchstaben auch eine Zahl entspricht, erweist sich in der Praxis in vielen Situationen als praktisch. Man denke beispielsweise an das Sortieren von Namen. Durch Zahlenvergleich können wir solche Aufgaben bewältigen.

Das folgende Beispiel ist kennzeichnend für die ASC- Funktion:

```
10 N=ASC("B")
20 PRINT N
```

Resultat:

```
66
```

Wir sehen, wie hier der ASCII-Wert von B durch die ASC- Funktion bestimmt wird.

**CHR$** Mit der CHR$-Funktion können wir wiederum das Zeichen bestimmen, das einem bestimmten ASC-Wert entspricht.
Beispiel:

```
PRINT CHR$(66)
```

Resultat:

```
B
```

Wenn wir diese beiden Beispiele miteinander vergleichen, wird klar, daß wir die CHR$-Funktion als eine Art 'umgekehrte' ASC-Funktion ansehen können.

VAL Mit Hilfe dieser Funktion können wir eine String, die eine Zahl angibt, auch wirklich in diese Zahl umwandeln.
Beispiel:

```
10 A$="123"
20 B=VAL(A$)
30 PRINT B
```

Resultat:

```
123
```

Denken Sie daran, daß A$ eine String-Variable ist. In Zeile 20 findet daher die eigentlich Umwandlung in die Zahl 123 statt.

**STR$** Die STR$-Funktion können wir als eine Umkehrung der VAL-Funktion auffassen: Wir verwandeln eine Zahl in eine String, die aus den gleichen Zeichen besteht.

Beispiel:

```
10 A=123
20 B$=STR$(A)
30 PRINT B$
```

Resultat:

```
123
```

Wir sehen, daß der Zahl-Variablen die Zahl 123 zugewiesen wird. Anschließend wird diese Zahl mit der STR$-Funktion der Variablen B$ als String zugewiesen.
Hierzu ist anzumerken, daß wir bei einer positiven Zahl in einer String immer einen Zwischenraum vor die String setzen.

**STRING$** Die Funktion STRING$ gebrauchen wir, um ein bestimmtes Zeichen, das als ASCII-Code angegeben ist, einige Male hintereinander zu setzen.
Die allgemeine Form ist:

```
STRING$ (K,ASCII-Wert)
```

Mit K geben wir an, wie oft das entsprechende Zeichen wiederholt werden soll. Wir können eventuell auch eine String angeben. Bei einer STRING$-Funktion wird dann das erste Zeichen der String betrachtet.
Beispiele:

```
PRINT STRING$ (5,65)        Resultat: AAAAA
PRINT STRING$ (5,"MSX")     Resultat: MMMMM
```

**SPACE$** Mit der SPACE$-Funktion können wir eine Anzahl Zwischenräume abdrucken.
Beispiel:

```
10 FOR K=1 TO 5
20 A$=SPACE$(K)
30 PRINT "A"; A$; "B"
40 NEXT K
```

Resultat:

```
A B
A  B
A   B
A    B
A     B
```

INSTR$(X$,Y$) Dies ist eine kraftvolle Such-Anweisung. Hiermit bestimmen wir nämlich, an welcher Stelle die String Y$ in der String X$ vorkommt.
Beispiel:

```
10 X$="WELTRAUM"
20 Y$="LTR"
30 K=INSTR(X$,Y$)
40 PRINT K
```

Resultat:

```
3
```

Die String LTR kommt ab der dritten Position in unserer Ausgangsstring WELTRAUM vor.

INKEY$ und ein Spielchen
INKEY$ ist keine Funktion, sondern eine Variable. Wenn der Computer bei der Verarbeitung eines Programmes auf den Ausdruck INKEY$ stößt, 'orientiert' er sich sofort an der Tastatur. Betätigt man in diesem Augenblick eine Taste, so wird das betreffende Zeichen INKEY$ zugewiesen. Wenn wir zu diesem Zeitpunkt keine Taste anschlagen, wird INKEY$ die leere String " " zugewiesen.
Das folgende Programm illustriert, wie INKEY$ für ein einfaches Reaktionsspiel verwendet werden kann.

```
10 PRINT "EINE TASTE ANSCHLAGEN"
20 PRINT "WENN DER BILDSCHIRM"
30 PRINT "DEN WERT 25 ZEIGT"
40 FOR K=1 TO 50
50 PRINT K
60 A$=INKEY$
70 IF A$ <> "" THEN GOTO 90
80 NEXT K
90 PRINT "ENDE"
```

Noch einige Bemerkungen Wir haben bisher die wichtigsten Funktionen hinsichtlich der Strings besprochen. Eine Übersicht aller zulässigen Anweisungen findet man im Anhang E.
Schließlich noch folgende Anmerkungen:

- Wie bei den Zahlen, dürfen wir auch String-Auflistungen dimensionieren, beispielsweise:

```
DIM A$(100)
```

- Der Computer hat beim Einschalten einen gewissen Teil des Speichers für die Speicherung von Strings reserviert. Als Standard können etwa 200 Zeichen gespeichert werden. Wir können diesen Raum mit der sogenannten CLEAR-Anweisung vergrößern, beispielsweise:

  ```
  CLEAR 500
  ```

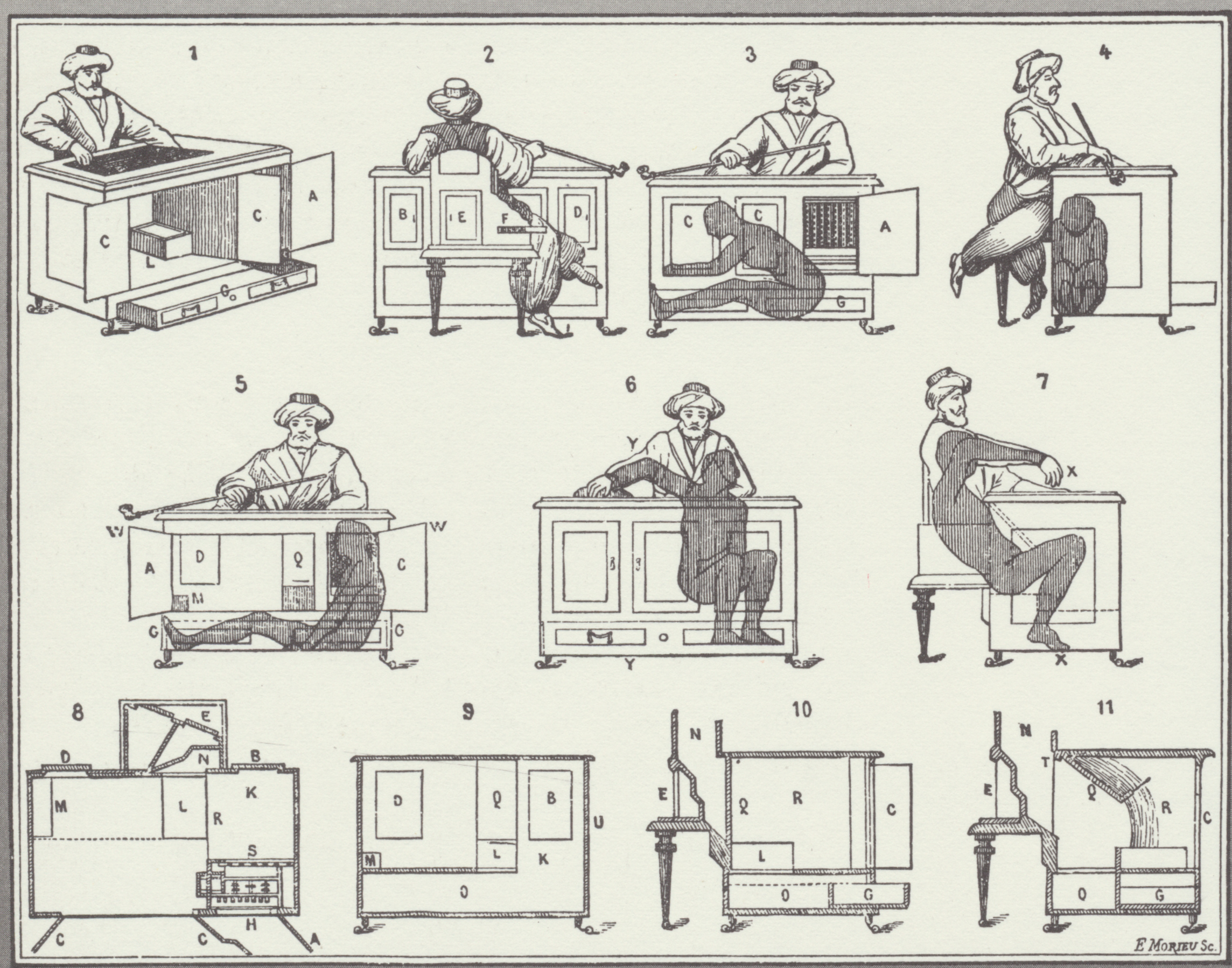
  Mit dieser CLEAR-Anweisung reservieren wir Platz zur Speicherung von 500 Zeichen.

- Die String, die wir einer String-Variablen zuweisen können, hat eine Höchstlänge von 255 Zeichen.

In diesem Kapitel kommen keine neuen Anweisungen zur Sprache. Wir werden unser bisher behandeltes Handwerkszeug zur Programmierung anwenden, um die Frage zu beantworten: 'Können Computer denken?' In diesem Kapitel werden daher zwei Programme besprochen.

Beim ersten Programm fordert uns der Computer zu einer Denksportübung heraus.

Beim zweiten Programm handelt es sich um das Erkennen eines Schemas. In unserem Beispiel handelt es sich um die Erkennung der Sprache eines formulierten Satzes.



13. Intermezzo: Können Computer denken?

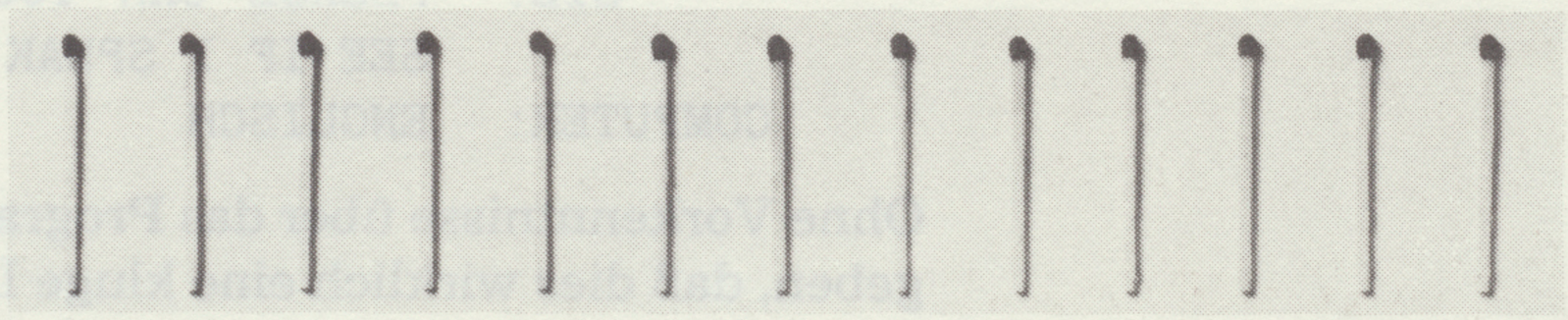
Einleitung

Die Frage 'Können Computer denken?' wird in vielen Artikeln gestellt. Diese Frage ist gar nicht so unsinnig, denn jeder von uns vergleicht einen Computer doch irgendwie mit dem menschlichen Gehirn. Was soll man schließlich von Schachprogrammen denken, die wir auf unserem Computer verarbeiten können. Sind solche Programme nicht ein Musterbeispiel für menschliches Denken?
Auf der Illustration sehen wir den berühmten türkischen Schachspieler, und wir wissen nachträglich, daß es sich dabei um Betrug handelte: Was dem Publikum als Schachautomat vorgeführt wurde, war in Wirklichkeit die raffinierte Verpakkung eines guten Spielers. Die Tatsache jedoch, daß uns derartige Automaten vorgesetzt werden, beweist immer wieder, wie sehr sich der Mensch darum bemüht, sein Denken in einen Automaten einzugeben.

Ein einfaches Denkspiel?

Um die Frage zu beantworten, ob Computer wirklich denken können, wollen wir etwas tiefer auf ein Denkspiel eingehen. Das Spiel, um das es sich hier handelt, ist das mit 13 Streichhölzern. Zwei Spieler dürfen abwechselnd 1, 2 oder 3 Streichhölzer wegnehmen oder - in diesem Fall - Sie und ihr Computer dürfen 1, 2 oder 3 Streichhölzer wegnehmen. Verlierer ist, wer das letzte Streichholz wegnehmen muß.
Sie selbst beginnen. Gesetzt den Fall, Sie nehmen 2 Streichhölzer und der Computer ebenfalls 2, so bleiben also noch 9 Streichhölzer übrig. Vorsichtig nehmen Sie nun 1 Streichholz, aber der Computer nimmt 3; so bleiben noch 5 Streichhölzer

Nochmals der berühmte schachspielende Mann (siehe auch Seite 30). Wie Sie sehen können handelte es sich nicht um einen Automaten... tatsächlich war im Schreibtisch ein Zettel.



übrig. Sie erkennen in diesem Augenblick, daß Sie verloren haben. Was immer Sie auch wegnehmen, der Computer nimmt solch eine Anzahl Streichhölzer, daß noch 1 Streichholz übrig bleibt: Nehmen Sie 2, so nimmt der Computer ebenfalls 2, und es bleibt noch 1 Streichholz für Sie übrig; nehmen Sie 1 Streichholz, so nimmt der Computer natürlich 3 usw. Sie sind der Verlierer, und wenn Sie den Trick nicht erkennen, macht der Computer natürlich den Eindruck, ein schlauer Denker zu sein. Bei einiger Nachdenklichkeit Ihrerseits erkennen Sie allerdings, daß der Computer die beiden ersten Male immer soviel Streichhölzer wegnimmt, daß 9 bzw. 5 Streichhölzer übrig bleiben. Anders ausgedrückt: die Summe der Streichhölzer, die Sie und der Computer wegnehmen, muß immer 4 sein.
Wenn man dies erkannt hat, ist es natürlich nicht mehr schwierig, ein Programm aufzustellen. Nehmen Sie etwa das folgende Programm, das seine Aufgabe ausgezeichnet erfüllt:

```
10 PRINT "SIE BEGINNEN"
20 INPUT K
30 PRINT "ICH NEHME DAVON..."; 4-K
40 PRINT "NUN SIND HIER NOCH 9 STREICHHÖLZER"
50 PRINT "NUN SIND SIE DRAN"
60 INPUT K
70 PRINT "ICH NEHME JETZT..."; 4-K
80 PRINT "JETZT SIND ES NUR NOCH 5 STREICHHÖLZER"
90 PRINT "SIE SIND WIEDER DRAN"
100 INPUT K
110 PRINT "ICH NEHME NATÜRLICH"; 4-K
120 PRINT "DA LIEGT NUR NOCH 1 STREICHHOLZ"
130 PRINT "HAHA ... SIE HABEN VERLOREN"
```

Sie werden vielleicht sagen, daß das nicht so nett ist, weil Sie den Trick durchschaut haben. Aber das ist doch immer so bei 'Gedankenarbeit', bei der wir genau angeben können, wie etwas geschehen muß. Betrachten wir daher das folgende Beispiel.

Das Erkennen deutscher und englischer Sätze

Bei diesem Beispiel soll der Computer feststellen, ob Sie einen englischen oder deutschen Satz eingeben.
Das Frage- und Antwortspiel des Computers verläuft folgendermaßen:

```
COMPUTER:  GEBEN SIE EINEN LANGEN SATZ IN
           DEUTSCH EIN (MIT GROSSBUCHSTABEN)
     SIE:  PLEASE CAN YOU TELL ME IF YOU CAN
           SEE IF I SPEAK ENGLISH
COMPUTER:  ENGLISCH
```

Ohne Vorkenntnisse über das Programm müssen Sie wohl zugeben, daß dies wirklich eine kluge Leistung ist. Hier gilt wie-

der: Wenn Sie nicht wissen, wie der Computer zu dem Ergebnis kommt, meinen Sie, der Computer sei eine intelligente Denkmaschine. Schauen wir uns aber mal an, wie der Computer zu dem Ergebnis kommt.
Ausgangspunkt ist, daß in längeren englischen Sätzen in der Regel der Buchstabe y vorkommt, während dies in deutschen Sätzen in der Regel nicht der Fall ist. Begreifen Sie den tieferen Sinn?
Wir müssen also nachschauen, ob in solch einem Satz der Buchstabe Y vorkommt.

Gehen wir einmal davon aus, daß das Programm folgende Anweisung enthält:

```
LINE INPUT A$
```

Dies bedeutet, der gesamte Satz wird als eine String oder eine Zeichenfolge A$ zugewiesen.
Wir müssen dann anschließend feststellen, ob der Buchstabe Y in dem Text vorkommt. Mit der folgenden Konstruktion können wir jeden Buchstaben für sich aus einem String herausziehen:

```
N=LEN(A$)
FOR K=1 TO N
B$=MID$(A$,K,1)
NEXT
```

Mit N wird die Anzahl der Zeichen gezählt, und die anschließende Konstruktion weist B$ nacheinander die einzelnen Zeichen der String A$ zu.
Zur Erprobung dieser Konstruktion kann man folgendes Programm eingeben:

```
10 LINE INPUT A$
20 N=LEN(A$)
30 FOR K=1 TO N
40 B$=MID$(A$,K.1)
50 PRINT B$
60 NEXT K
```

Wenn wir jetzt beispielsweise COMPUTER tippen, sehen wir als Ergebnis:

```
C
O
M
P
U
T
E
R
```

Jetzt müssen wir noch dafür sorgen, daß der Computer jedes Zeichen betrachtet, und wenn dies tatsächlich ein Y ist, muß die Erklärung 'ENGLISCH' folgen.
Kurz und gut, wir benötigen folgende Anweisung in unserem Programm:

```
IF B$="Y" THEN PRINT "ENGLISCH"
```

In diesem Fall ist es praktisch, nach der PRINT- Anweisung auch die STOP-Anweisung einzugeben, denn wenn wir erst einmal wissen, daß es sich um einen englischen Satz handelt, braucht der Computer nicht weiter zu suchen. Hier folgt unser vollständiges Programm:

```
10 PRINT "SCHREIBEN SIE EINEN LANGEN DEUTSCHEN ODER
ENGLISCHEN SATZ (IN GROSSBUCHSTABEN)
20 LINE INPUT A$
30 N=LEN(A$)
40 FOR K=1 TO N
50 B$=MID$(A$,K.1)
60 IF B$="Y" THEN PRINT "ENGLISCH": STOP
70 NEXT K
80 PRINT "DEUTSCH"
```

Wieder werden Sie nachträglich den Eindruck gewonnen haben, daß Sie etwas genasführt worden sind - es war schließlich furchtbar einfach.
Doch dieses Programm ist recht beispielhaft, weil es uns zeigt, wie wir mit der IF...THEN-Anweisung den Computer zum 'Nachdenken' bringen können.

Mustererkennung

Nach der Behandlung dieses Programmes ist es nochmals sinnvoll, die Frage zu stellen, ob Computer 'denken' können. Es ist wohl klar, daß Computer denken können, wenn wir den Begiff 'denken' in die Verarbeitung einer Reihe von BASIC-Anweisungen übersetzen können. In vielen Fällen handelt es sich bei diesem Denken um etwas, das man auch mit dem Wort 'Mustererkennung' kennzeichnen kann. Bei der Mustererkennung handelt es sich schließlich um die Feststellung, ob ein bestimmtes Muster vorhanden ist oder nicht. Zur Durchführung der dafür notwendigen Untersuchung(en) müssen natürlich wir selbst die Kennzeichen dieses sogenannten Musters kennen. So war eines der Kennzeichen eines englischen Satzes, daß der Buchstabe Y relativ häufig vorkommt.
Wir müssen uns bewußt werden, daß wir selbst auch von derartigen Untersuchungsmechanismen ausgehen. In unserer Schulzeit mußten wir uns allerlei englische Wörter wie YOU, PLEASE, TELL ME und ENGLISH einprägen. Wenn wir einen beliebigen Satz hören, erkennen wir diese Wörter sofort. An-

ders gesagt: unser Erkennungsmechanismus stellt sofort Übereinstimmungen fest zwischen den Wörtern in einem Satz und Wörtern, die wir zuvor gelernt haben.
Es gibt unzweifelhaft starke Übereinstimmungen zwischen dem menschlichen Denken und dem 'Denken' des Computers. Dabei müssen wir allerdings begreifen, daß die in unserem Gehirn ablaufenden 'Programme' uns weitgehend unbekannt sind. Hieraus ergibt sich wiederum, daß eine wirklich richtige Beantwortung unserer Ausgangsfrage wahrscheinlich nie gefunden werden kann.

Dessen ungeachtet können wir behaupten, daß Computer uns um so mehr zum Nachdenken über unser eigenes Denken zwingen, und dies ist zumindest eine wesentliche Feststellung!

1 Was ist eine Mustererkennung?
2 WAS IST EINE MUSTERERKENNUNG?
3 Was ist eine Mustererkennung?
4 Was ist eine Mustererkennung?
5 Was ist eine Mustererkennung?
6 [illegible] ist eine Mustererkennung?
7 Was ist eine Mustererkennung?

Hier oben sehen wir den Satz 'Was ist eine Mustererkennung?' siebenmal geschrieben.
Auf besonders deutliche Weise sehen wir hier die eigentlichen Probleme der Mustererkennung. Wir Menschen sind mit einem besonders guten Mustererkennungsmechanismus ausgestattet. Wenn wir derartige Sätze von einem Computer identifizieren lassen wollen, müßte jeder einzelne Buchstabe zuerst abgesondert werden. Bei Zeile 2 und 7 ist das vielleicht möglich, aber wie ist es beispielsweise mit Zeile 6?
Die Erkennung handgeschriebener Sätze durch einen Computer ist jetzt bereits ein geradezu klassisches Problem der Mustererkennung!

In diesem Kapitel wird die Subroutine (Unterprogramm) behandelt. Eine Subroutine ist faktisch eine Reihe von Anweisungen, die aus einem beliebigen Teil unseres 'Hauptprogrammes' abgerufen werden kann, als wäre sie ein gesondertes Programm. Der wesentlichste Gewinn von Subroutinen ist es, daß wir damit die Möglichkeit zu einem strukturierten Programmaufbau bekommen.
Dieser Punkt ist daher auch der wesentlichste Bestandteil dieses Kapitels. Wir werden hierbei unter anderem sehen, wie wir ein Problem in Teilprobleme aufteilen können, die ihrerseits auch wieder aufgeteilt werden können. Diese Methode ist auch bekannt unter der Bezeichnung 'Methode der schrittweisen Verfeinerung'. Schließlich werden wir sehen, wie wir mit Hilfe der DEF FN-Anweisung selbst Funktionen definieren können.

14. Subroutinen (Unterprogramme) und mehr über das Entwickeln eines Programms

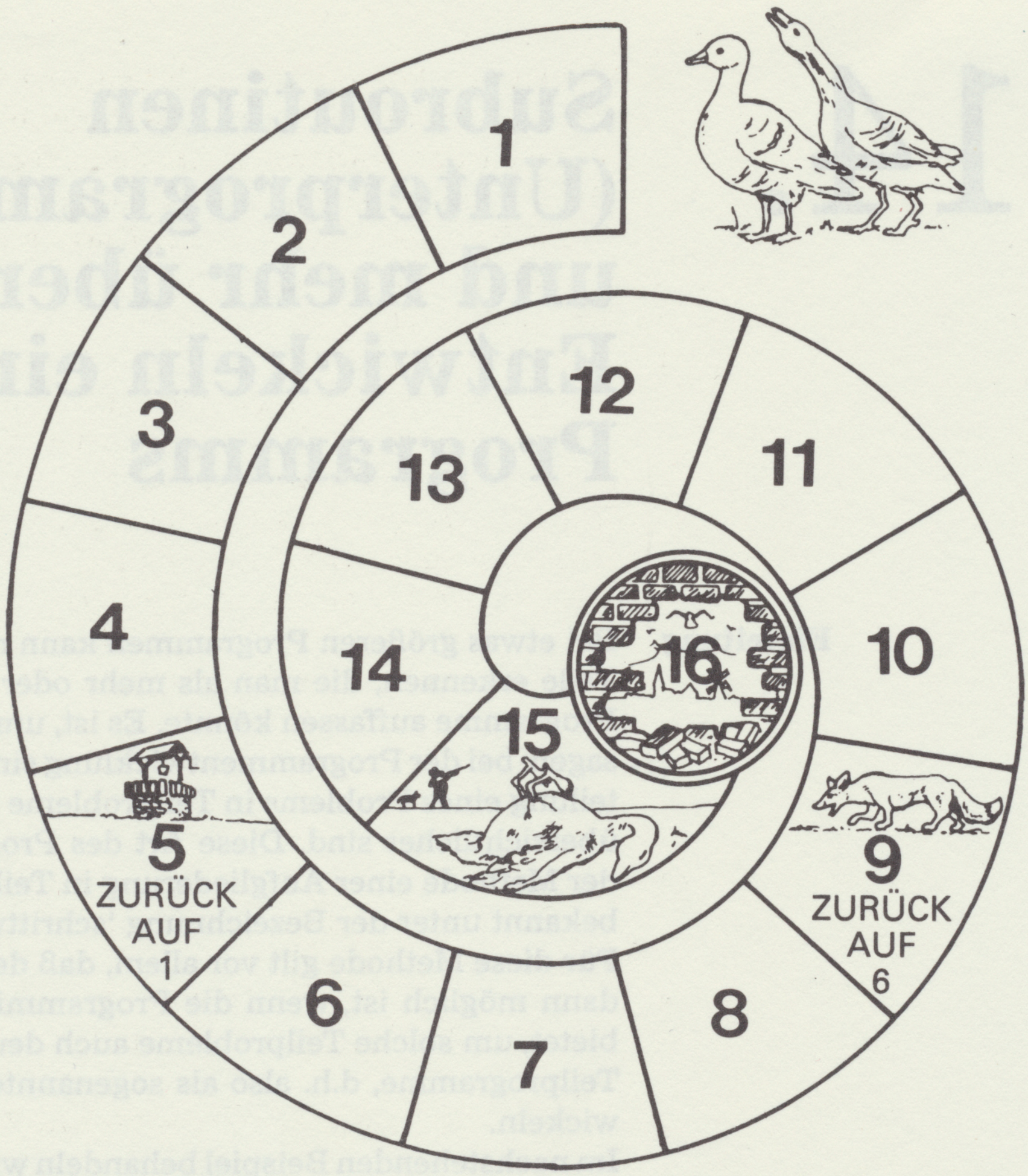
Einleitung

Bei etwas größeren Programmen kann man im Aufbau häufig Teile erkennen, die man als mehr oder weniger selbständige Programme auffassen könnte. Es ist, um es noch deutlicher zu sagen, bei der Programmentwicklung sinnvoll, nach der Unterteilung eines Problems in Teilprobleme zu suchen, die an sich übersichtlicher sind. Diese Art des Programmierens, die von der Methode einer Aufgliederung in Teilprobleme ausgeht, ist bekannt unter der Bezeichnung 'schrittweise Verfeinerung'.
Für diese Methode gilt vor allem, daß der richtige Aufbau nur dann möglich ist, wenn die Programmiersprache Werkzeuge bietet, um solche Teilprobleme auch deutlich als selbständige Teilprogramme, d.h. also als sogenannte Subroutinen zu entwickeln.
Im nachstehenden Beispiel behandeln wir den Aufbau eines etwas größeren Programmes. Hierbei wird das ursprüngliche Problem in kleinere, logische Teilprobleme unterteilt werden. Beim Umsetzen dieser Teilprobleme in selbständige Teilprogramme machen wir von der Subroutine-Struktur Gebrauch, die MSX-BASIC bietet.

Das Gänsespiel

Das von uns jetzt aufzustellende Programm bietet uns die Möglichkeit, das altholländische Gänsespiel zu spielen.
Der Übersichtlichkeit wegen haben wir unser Gänsespielfeld etwas vereinfacht gehalten. Zur Übung kann man danach die hier besprochene Technik anwenden, um ein Programm für das originale Gänsespielbrett zu schreiben.
Die Abbildung auf der nächsten Seite zeigt unser Gänsespielbrett.

Der zurückzulegende Weg besteht aus 16 Feldern. Die Spieler beginnen auf Feld 1 und würfeln der Reihe nach mit einem Würfel. Die gewürfelten Augen bestimmen, wieviele Schritte man gehen darf.



Verschiedene Felder haben, wie das beim Gänsespiel der Fall ist, eine besondere Bedeutung:

Feld 5: Wer hier kommt, muß wieder neu anfangen und damit auf Feld 1 zurück.

Feld 9: Zurück auf Feld 6.

Feld 15: Dies ist die bekannte Falle vor Ende des Spiels. Wenn man hier landet, muß man wieder erneut anfangen, also zurück auf Feld 1.

Es kommt darauf an, auf Feld 16 zu kommen. Dies kann man nur, wenn man zuletzt genau die richtige Augenzahl würfelt. Zeigt der Würfel eine zu große Anzahl, muß man entsprechend zurückzählen. Ist man beispielsweise auf Feld 14 und würfelt 4, dann würde man auf 14+4=18 kommen, doch nun wird wieder zurückgezählt, und man gelangt erneut auf Feld 14.

Programm Wir gehen davon aus, daß es zwei Spieler gibt: Sie selbst und der Computer. Dem Stand beider Spieler, also dem Feld, auf dem

Sie stehen, können wir natürlich eine Variable zuweisen. Wir gebrauchen beispielsweise die Variable A, um Ihren Stand anzugeben und Variable B für den Stand des Computers. Das Spiel könnte man dann folgendermaßen beschreiben:

1 A und B den Startwert 1 geben. Dies sind die Startpositionen.
2 Der Spieler ist an der Reihe.
3 Der Computer ist an der Reihe.
4 Zurück zu Schritt 2.

Wir sehen, wie hier das ursprüngliche Problem in vier Teilprobleme unterteilt ist. Nun untersuchen wir, wie das Teilproblem 2 weiter aufgeteilt werden kann.

2a Würfeln. Wir gehen davon aus, daß der gewürfelten Augenzahl die Variable D zugewiesen wird.
b Das neue Feld berechnen: A=A+D
c Prüfen, ob das neue Feld ein besonderes Feld ist und nötigenfalls das neue Feld korrigieren. Wenn A jetzt z.B. 9 würde, wird A gemäß den Spielregeln korrigiert (A wird dann 6).

Hierzu ist anzumerken, daß die Ausarbeitung von Schritt 3 - der Computer ist an der Reihe - zu einer gleichen Aufgliederung führt.

3a Würfeln. Wir gehen davon aus, daß der gewürfelten Augenzahl die Variable D zugewiesen wird.
b Das neue Feld berechnen: B=B+D.
c Feststellen, ob das erwürfelte Feld ein besonderes Feld ist und nötigenfalls das neue Feld korrigieren.

Die Schritte a und c der Teilprobleme 2 und 3 sind völlig identisch, und es wäre daher angenehm, wenn wir hierfür den gleichen Programmteil verwenden könnten.

Wir werden nun ein 'BASIC-artiges' Programm aufschreiben, bei dem die einzelnen Subroutinen noch in Worten notiert werden.

```
A=1 }
B=1 }  Startposition
PRINT "SIE SIND DRAN"
```

Subroutine TASTE

Subroutine aufrufen, in der der Spieler eine beliebige Taste betätigen muß. Dies hat den Effekt, daß der Spieler etwas tun muß.

Subroutine WÜRFELN

Subroutine aufrufen, um die Augenzahl von Würfel D zu bestimmen.

```
A=A+D
```

Subroutine KORREKTUR

Feststellen, ob es sich um ein besonderes Feld handelt und erforderlichenfalls den Stand korrigieren. Neuer Stand ist N.

```
A=N
PRINT "IHR NEUER STAND IST"; N
PRINT "COMPUTER IST DRAN"
```

Subroutine WÜRFELN

Subroutine aufrufen, um die Augenzahl von Würfel D zu bestimmen

```
B=B+D
```

Subroutine KORREKTUR?

Feststellen, ob es sich um ein besonderes Feld handelt. Neuer Stand ist N.

```
B=N
PRINT "NEUER STAND DES COMPUTERS IST"; N
```

Zurück zur Anweisung: `PRINT "SIE SIND DRAN"`

GOSUB...RETURN

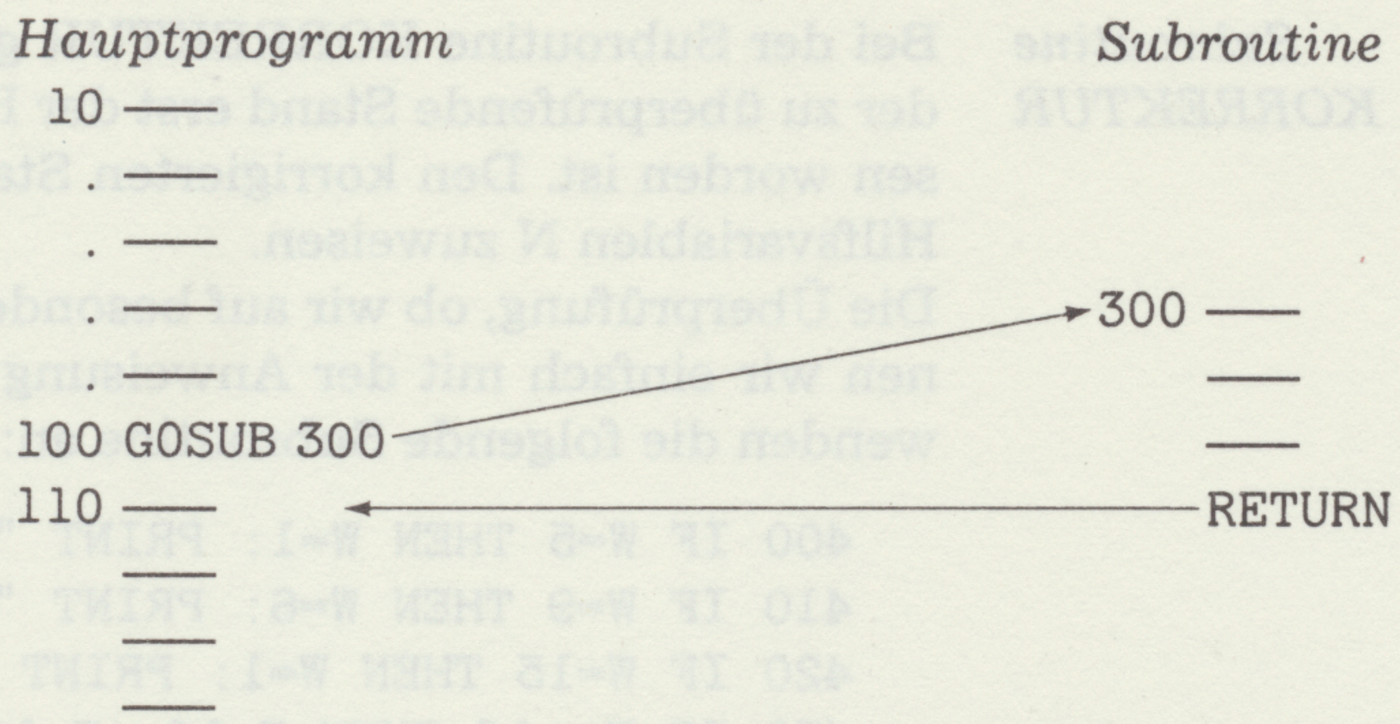
Wir sind nun an dem Punkt angelangt, das 'Pseudoprogramm' in ein echtes Programm umzusetzen.
Bevor wir dies tun, wollen wir zuerst die Subroutine-Konstruktion von MSX-BASIC darstellen.
Diese Konstruktion bedeutet, daß wir zu jedem beliebigen Programmteil springen können, wenn wir folgende Anweisung gebrauchen:

`GOSUB` Zeilennummer

und daß der Computer nach dem Erkennen des Ausdrucks

`RETURN`

in diesem Programmteil wieder zum ursprünglichen Programmteil zurückspringt. Der Computer setzt das Programm dann mit der Anweisung fort, die in der Zeile nach der GOSUB-Anweisung steht.
Siehe Schema auf der folgenden Seite.
In diesem Schema ist der Übersichtlichkeit halber rechts vom 'Hauptprogramm' die Subroutine angegeben. Die Programme würden in Wirklichkeit eine lange Liste ergeben.



Wir besprechen nun nacheinander die Subroutinen unseres Gänsespiel-Programmes.

Subroutine TASTE

In dieser Subroutine wollen wir den Effekt einer Unterbrechung aufrufen. Der Spieler muß zuerst eine Taste betätigen, um gewissermaßen zu würfeln.
Wir machen hier natürlich wieder Gebrauch von der INKEY$-Konstruktion. Wir wenden folgende Subroutine an:

```
200 PRINT "EINE TASTE BETÄTIGEN"
210 A$=INKEY$
220 IF A$="" THEN GOTO 210
230 RETURN
```

Subroutine WÜRFEL

Diese Subroutine bewirkt eine Würfelimitation.
Hierbei wenden wir die sogenannte RND-Funktion an, die folgende Form hat*:

```
RND(1)
```

Diese Funktion ergibt eine beliebige Zahl zwischen 0 und 1. Um jedoch eine beliebige Zahl zwischen 1 und 6 zu erhalten, wenden wir folgende Konstruktion an:

```
INT(RND(1)*6+1)
```

Die Würfel-Subroutine wird dann:

```
300 D=INT(RND(1)*6+1)
310 PRINT "DER WÜRFEL ZEIGT EINS"; D
320 RETURN
```

*Anmerkung: Um wirklich nicht vorauszusagende Ergebnisse zu erzielen, geht man aus von RND(_TIME).

Subroutine KORREKTUR

Bei der Subroutine KORREKTUR gehen wir davon aus, daß der zu überprüfende Stand erst der Hilfsvariablen W zugewiesen worden ist. Den korrigierten Stand werden wir dann der Hilfsvariablen N zuweisen.
Die Überprüfung, ob wir auf besonderen Feldern stehen, können wir einfach mit der Anweisung IF...THEN angeben. Wir wenden die folgende Subroutine an:

```
400 IF W=5 THEN W=1: PRINT "ÜBER FELD 5"
410 IF W=9 THEN W=6: PRINT "ÜBER FELD 9"
420 IF W=15 THEN W=1: PRINT "ÜBER FELD 15"
430 IF W=>16 THEN W=16-(W-16): PRINT
"ZU WEIT....": GOTO 400
440 IF W=16 THEN PRINT "GEWONNEN!": STOP
450 N=W
460 REM WARTEZEIT
470 FOR K=1 TO 400
480 NEXT K
490 RETURN
```

Die Zeilen 400 bis 450 sprechen für sich. Die PRINT-Anweisungen wurden hinzugefügt, damit die Spieler erkennen, wie sie das neue Feld erreicht haben. Der letzte Teil wurde hinzugefügt, um eine gewisse Wartezeit zu verursachen. Damit sieht es ganz so aus, als müßten die Figuren versetzt werden...

Vollständiges Programm

Nach der obigen Darlegung wollen wir nun das komplette Programm darstellen. Wir werden sehen, daß wir uns völlig an unsere ursprüngliche Planung gehalten haben.
Gänsespiel:

```
10  A=1
20  B=1
30  PRINT "SIE SIND DRAN"
40  GOSUB 200
50  GOSUB 300
60  A=A+D
70  W=A
80  GOSUB 400
90  A=N: PRINT "IHR NEUER STAND IST"; N:PRINT
100 PRINT "COMPUTER IST DRAN"
110 GOSUB 300
120 B=B+D
130 W=B
140 GOSUB 400
150 B=N: PRINT "NEUER STAND COMPUTER IST";
    N:PRINT
160 GOTO 30
```

```
200 PRINT "TASTE BETÄTIGEN"
210 A$=INKEY$
220 IF A$="" THEN GOTO 210
230 RETURN
300 D=INT (RND(1)*6+1)
310 PRINT "WÜRFEL ZEIGT EINS"; D
320 RETURN
400 IF W=5 THEN W=1: PRINT "ÜBER FELD 5"
410 IF W=9 THEN W=6: PRINT "ÜBER FELD 9"
420 IF W=15 THEN W=1: PRINT "ÜBER FELD 15"
430 IF W=>16 THEN W=16-(W-16): PRINT
    "ZU VIEL....": GOTO 400
440 IF W=16 THEN PRINT "GEWONNEN!": STOP
450 N=W
460 REM WARTEZEIT
470 FOR K=1 TO 400
480 NEXT K
490 RETURN
```

Nachbetrachtung

Nach dieser ersten Besprechung der Subroutine wollen wir uns nochmals mit den wesentlichen Vorteilen dieser Subroutine-Konstruktion befassen. In vielen Textbüchern wird darauf hingewiesen, daß die Subroutine vor allem vorteilhaft ist, wenn wir eine bestimmte Serie von Anweisungen mehrmals verarbeiten müssen.
Das stimmt natürlich, doch unseres Erachtens liegt der Wert der Subroutine hauptsächlich darin, daß wir bei der Programmaufstellung die mehr oder weniger gesonderten Teile als selbständige Teilprogramme entwickeln können.
Das Hauptprogramm besteht dann aus einer Reihe von Anweisungen, in denen unter anderem die Subroutinen aufgerufen werden. So erhält man folgende Struktur:

```
10 REM HAUPTPROGRAMM
   ____
   ____
   GOSUB-----
   ____
   GOSUB-----
   usw.
   ____________

   ____
   ____
   ____

   ____
   ____
   ____
```

Hauptprogramm

Subroutine 1

Subroutine 2

Ein so aufgestelltes Programm bietet eine deutliche Struktur, und das ist eigentlich der Hauptzweck eines guten Programmes!

Schließlich noch folgende Anmerkungen:

- Subroutinen selbst können auch wieder Subroutinen abrufen.
- Die ON...GOTO-Anweisung ist ebenso wie die ON...GOSUB-Anweisung einzusetzen. Die Aufstellung ist völlig übereinstimmend mit der ON...GOTO- Anweisung, und darum wollen wir hierauf nicht weiter eingehen.

DEF FN MSX-BASIC bietet zugleich die Möglichkeit, selbst Funktionen zu definieren. Solch eine Funktion können wir als eine Art von Subroutine auffassen, und aus diesem Grund besprechen wir diese Möglichkeit auch in diesem Kapitel.
Die DEF FN-Anweisung hat immer folgende Form:
DEF FN (Reihe Variablen, getrennt durch Kommas)= Ausdruck, worin diese Variablen vorkommen.
Beispiel:

```
DEF FNA (X,Y)=X∧2+Y∧2
```

Der Funktionsname besteht aus nur einem Buchstaben (in unserem Fall dem A). In diesem Beispiel wird die Funktion A definiert durch:
X^2+Y^2
Diese Funktion wird dann aufgerufen mit dem Ausdruck FNA, d.h. mit dem Ausdruck FN, gefolgt durch den Namen der Funktion.
Folgendes Programm macht dies deutlich:

```
10 DEF FNA(X,Y)=X∧2+Y∧2
20 A=3
30 B=4
40 C=FNA(A,B)
50 PRINT C
```

Resultat:

```
25
```

Zeile 10 definiert auf besprochene Weise die Funktion. In erster Instanz negiert der Computer diese Anweisung. Zeile 20 und 30 sprechen für sich. Zeile 40 zeigt den Ausdruck FNA. Der Computer leitet hieraus ab, daß eine Funktion aufgerufen wird, und zwar die Funktion mit dem Namen A. Anschließend schaut er nach Zeile 10, und überall dort, wo dort X steht, wird A eingesetzt, und auf gleiche Weise wird für Y die Variable B eingesetzt. Auf diese Weise wird C:

A∧2+B∧2

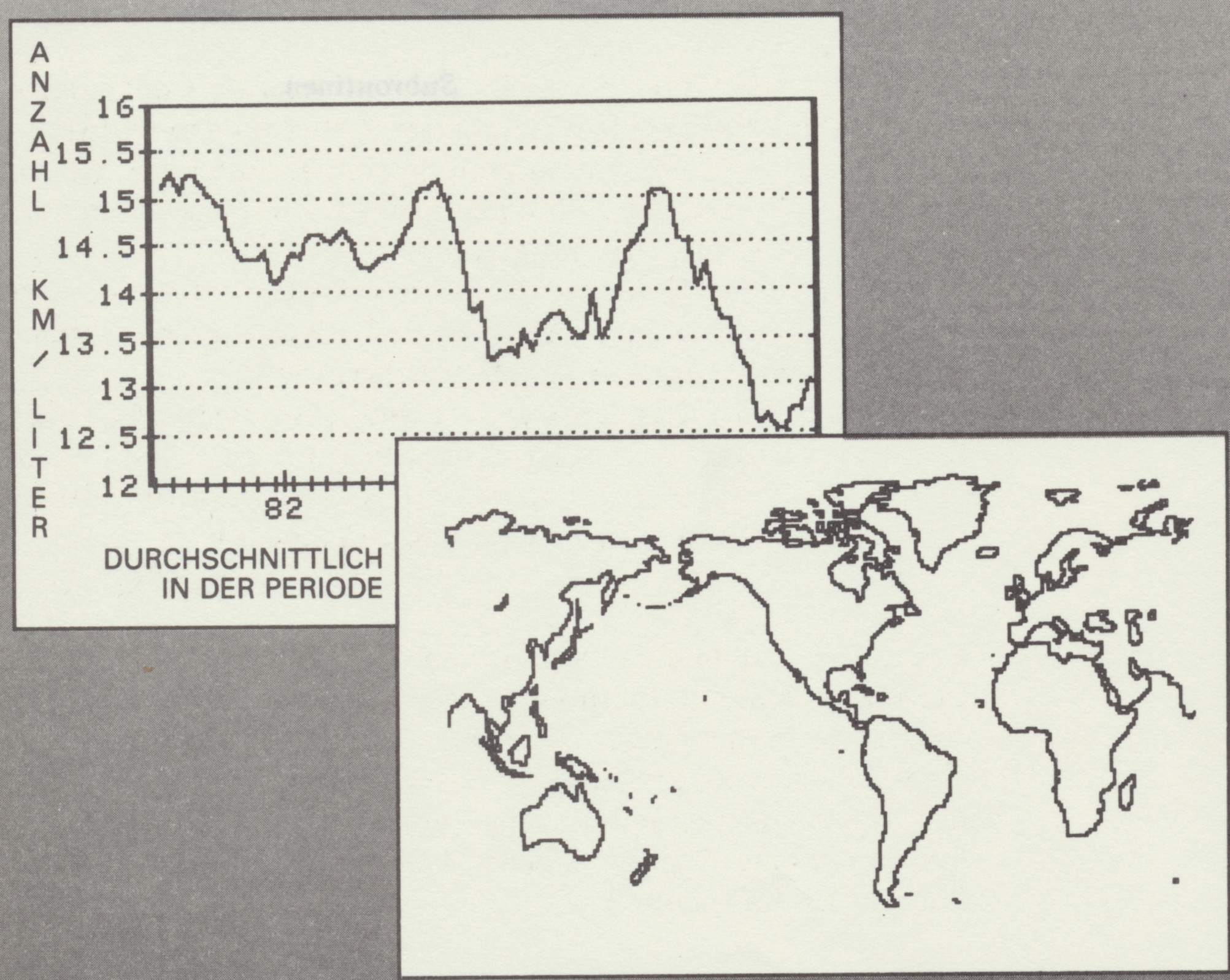
und da A=3 und B=4 ist, wird C:

3∧2+4∧2 $(=3^2+4^2)$

Dies wird durch das Resultat bestätigt.

Subroutinen...

Die auszuführende Information der bisher behandelten Programme bestand aus Text und Zahlen. In diesem Kapitel lernen wir die Möglichkeit kennen, auch allerlei grafische Darstellungen auf dem Bildschirm wiederzugeben.
Wir besprechen die SCREEN-Anweisung, um den sogenannten grafischen Bildschirm zu definieren, die PSET-Anweisung, um einen Punkt auf dem Bildschirm hervorzurufen, COLOR zur Wiedergabe von Farbe und PRESET, um einen bereits angegebenen Punkt wieder zu löschen.



15. Grafische Darstellungen: SCREEN, PSET, COLOR und PRESET

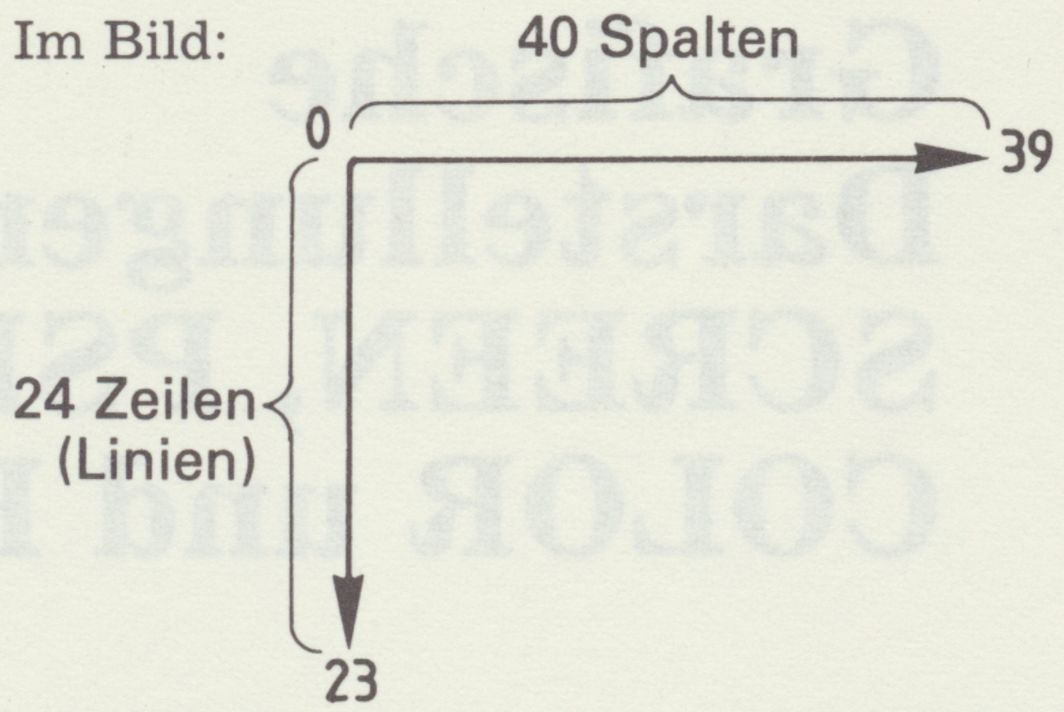
Einleitung

Eine grafische Darstellung ist nichts anderes, als ein Ausdruck für eine Abbildung. Abbildungen bauen wir auf aus Punkten, Linien und Kurven.
Unser MSX-Computer hat besonders viele Möglichkeiten, allerlei hübsche Figuren auf den Bildschirm zu bringen. Das ist wichtig, denn viele Einsatzbereiche sind gerade an das Arbeiten mit packenden Bildern gebunden. Ein illustratives Beispiel wären beispielsweise allerlei Bildungsprogramme. Themen, wie etwa das Rechnen, werden bei diesen Programmen mit eindrucksvollen Darstellungen auf dem Bildschirm verbunden, um das Lernen zu erleichtern.
Es besteht wohl kein Zweifel darüber, daß sich Computer in diesem Bereich einen unersetzbaren Platz erobern werden.
Abgesehen von den Bildungsprogrammen müssen wir uns auch, wenn wir an Abbildungen denken, die vielen angebotenen Computerspiele vor Augen halten.
Trotz deutlicher Gefahren einer 'Süchtigkeit' (bei gewissen Spielen) läßt sich nicht verleugnen, daß wir mitunter besonders ansprechende Produkte zu sehen bekommen.
Es führt unseres Erachtens viel zu weit, alles, was mit Computerspielen zu tun hat, einfach zu verurteilen. Die persönliche Erfahrung lehrt, daß eigenes Schreiben eines Spieles eine sehr fesselnde und nicht weniger belehrende Erfahrung ist.

SCREEN

Um einen Einblick zu bekommen, wie wir mit einem MSX-Computer Bilddarstellungen aufbauen können, müssen wir zuerst etwas mehr über die Einteilung unseres Bildschirmes (englisch: screen) erzählen.
Bei all den bisher gebotenen Darstellungen gebrauchte der Computer eine Einteilung von 24 Zeilen, wobei jede Zeile 40 Zeichen enthalten kann.

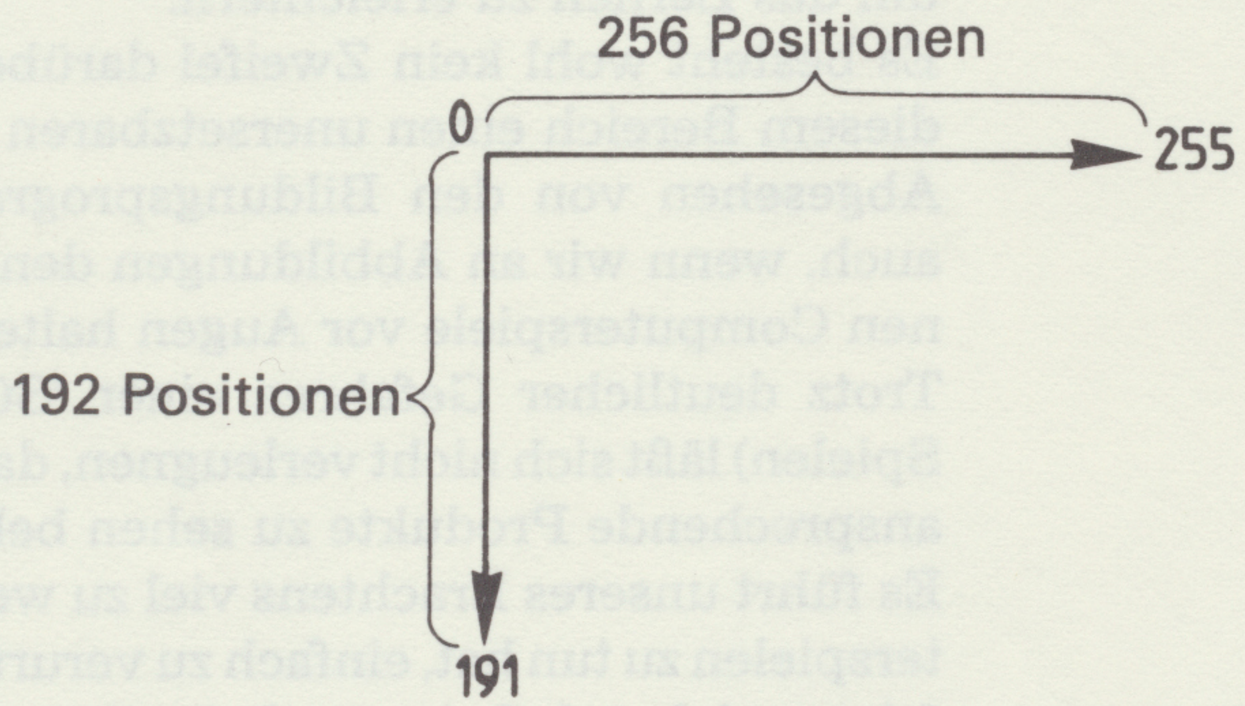
Im Bild:



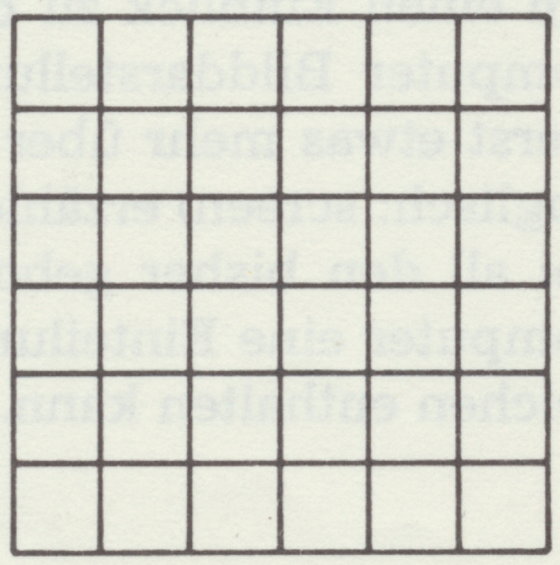
Wie Sie merken, ist hier das Achsensystem mehr oder weniger im Kopfstand gezeichnet. Bei Computern ist das so üblich.
Wenn wir feine Zeichnungen wiedergeben wollen, ist diese Aufteilung unzureichend. Wenn wir beispielsweise auf einem Schirm mit 24x40 Positionen mit Kreuzchen eine Linie zeichnen wollen, so ergibt sich eine recht grobe Figur.
Um dennoch Linien mit hoher Feinheit (in Prospekten spricht man über hohe Auflösung) darstellen zu können, müssen wir den Computer zuerst umschalten, so daß eine andere Bildschirmeinteilung erreicht wird. Nach der Anweisung:

```
SCREEN 2
```

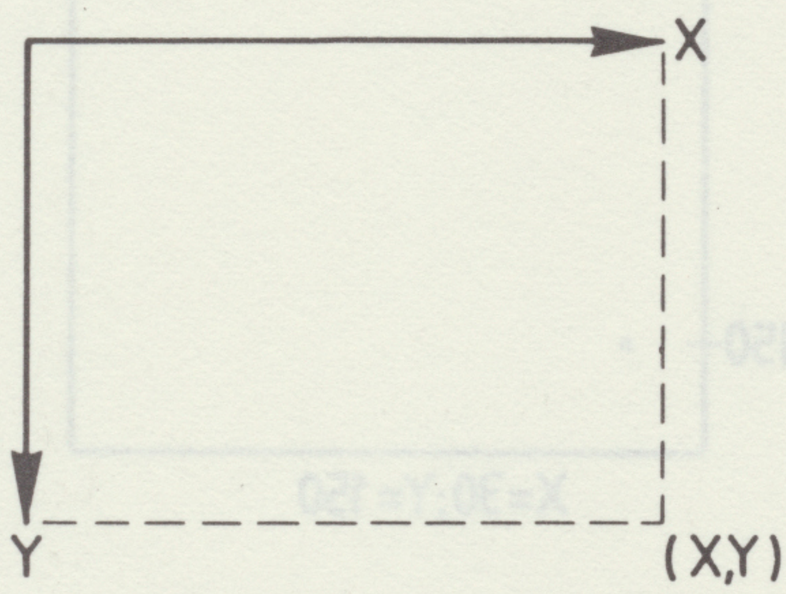
bringt der Computer folgende Aufteilung zustande:



Wir erkennen nun eine viel feinere Rasterung unseres Bildschirmes. Betrachten wir wiederum einen Teil dieses Rasters, so erkennen wir ein System von Quadraten.



Jedes dieser Quadrate bezeichnen wir von jetzt an als Bildpunkt (oder kurz Punkt), und die Koordinaten jedes Punktes geben wir dann durch einen X- und einen Y- Wert an gemäß der folgenden Regel:



Hierbei darf x von 0 bis 255 und y von 0 bis 191 variieren. Punkt (0,0) liegt daher in dem Punkt links oben, sowie Punkt (255,191) unten rechts.
Nachdem wir mit SCREEN 2 den Computer zuerst einmal in die Lage versetzt haben, das feine Raster zu nutzen, können wir die Punkte und Linien mit dazu verfügbaren Funktionen angeben.

PSET Wir beginnen mit der Anweisung PSET (=pointset, d.h. Wiedergabe eines Punktes), so daß wir Punkte auf dem Bildschirm angeben können.
In der einfachsten Form sieht diese Anweisung wie folgt aus:
PSET (x-Koordinate, y-Koordinate)

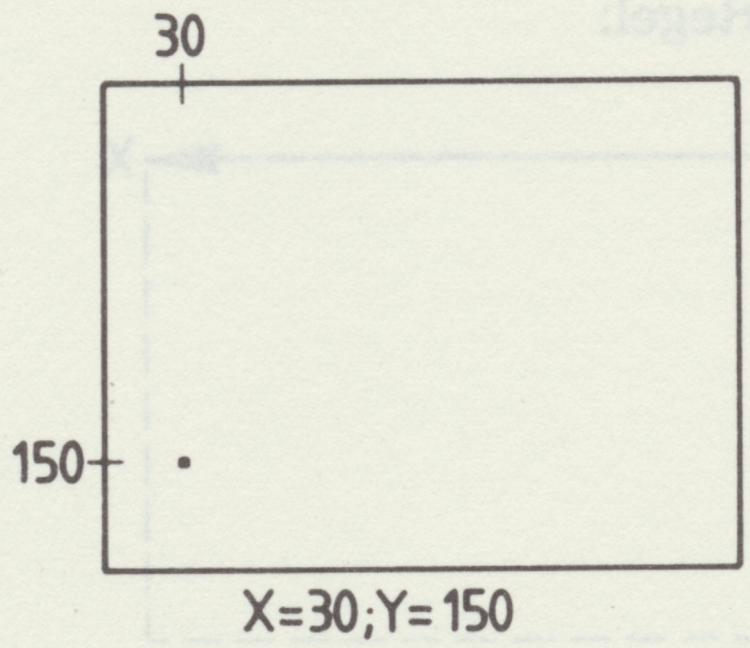
Wir beginnen sofort mit einem Beispiel:

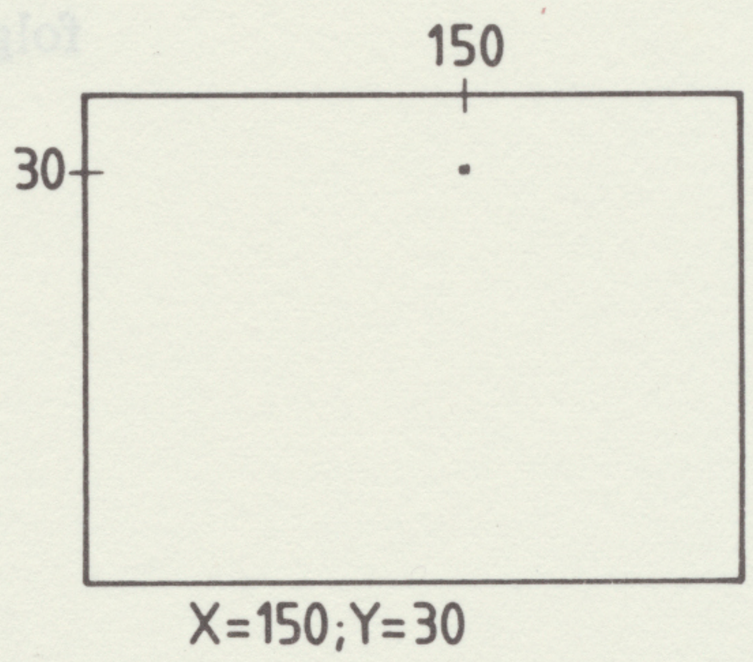
```
10 INPUT "X=";X
20 INPUT "Y=";Y
30 SCREEN 2
40 PSET (X,Y)
50 GOTO 50
```

Mit Zeile 10 und 20 weisen wir X und Y einen Wert zu. In Zeile 30 steht die notwendige Anweisung SCREEN 2. Danach wird der Punkt mit der PSET-Anweisung gezeichnet. Die letzte Anweisung sorgt dafür, daß das Programm nicht beendet wird, so daß wir faktisch nichts zu sehen bekämen (lassen Sie diese Zeile versuchshalber einmal weg!).
Wir können das Programm daher auch nur unterbrechen, indem wir mit betätigter CTRL-Taste auf STOP drücken (CTRL/STOP).

Geben Sie nun einmal einige Werte ein. Beachten Sie, daß sich durch kleine Werte Punkte ergeben können, die außerhalb des

Bildschirmfeldes liegen, so daß wir derartige Punkte nicht zu sehen bekommen. Nachstehende Skizzen zeigen zwei Beispiele:





Unsere PSET-Anweisung können wir noch dadurch erweitern, daß wir eine Zahl dahintersetzen, die die Farbe angibt. Verändern Sie einmal das Programm auf folgende Weise:

```
10 INPUT "X=";X
20 INPUT "Y=";Y
30 INPUT "FARBE=";K
40 SCREEN 2
50 PSET (X,Y),K
60 GOTO 60
```

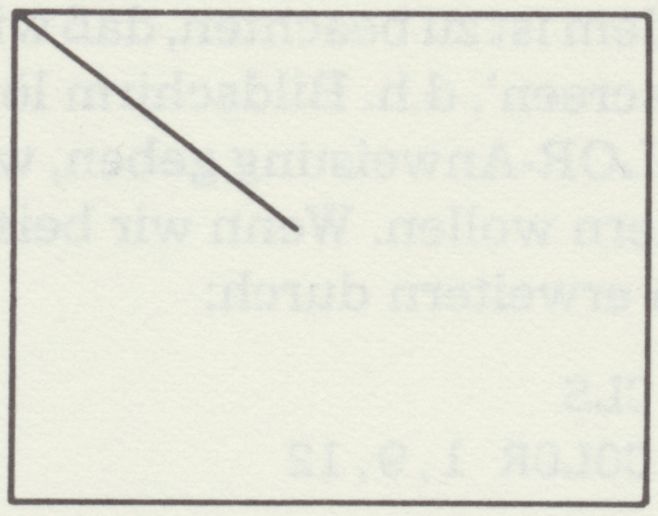
Für die Farben gelten folgende Zahlen:

Zahl/Farbe	*Zahl/Farbe*	*Zahl/Farbe*	*Zahl/Farbe*
0 transparent	4 dunkelblau	8 rot	12 dunkelgrün
1 schwarz	5 hellblau	9 hellrot	13 magenta
2 grün	6 dunkelrot	10 dunkelgelb	14 grau
3 hellgrün	7 himmelblau	11 hellgelb	15 weiß

In einigen Fällen werden Sie den farbigen Punkt nicht deutlich oder überhaupt nicht erkennen. Wenn wir eine Reihe von Punkten setzen, ist der Effekt jedoch bestimmt deutlicher. Folgendes Programm bietet dazu die Möglichkeit:

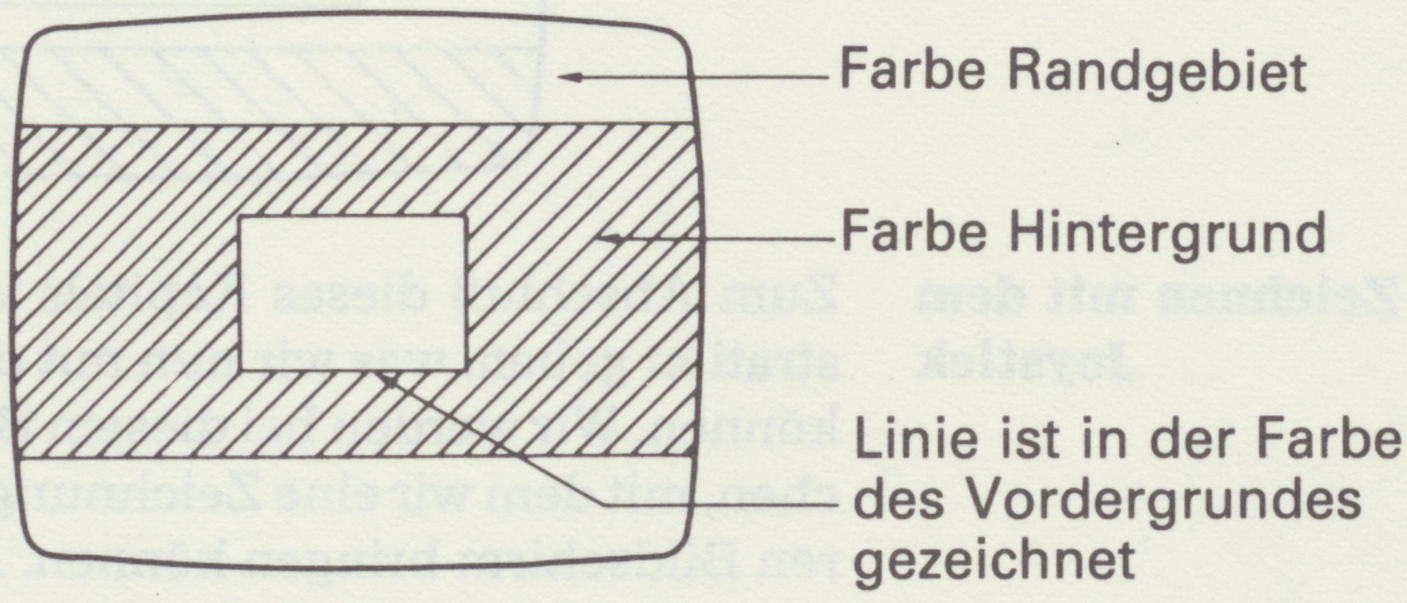
```
10 INPUT "N=";N
20 INPUT "FARBE=";K
30 SCREEN 2
40 FOR J=1 TO N
50 PSET (J,J),K
60 NEXT J
70 GOTO 70
```

Wenn wir nun beispielsweise für N den Wert 100 eingeben und für K den Wert 1, sehen wir folgende schwarze Linie:



Dieses Beispiel zeigt zugleich, daß wir Linien zeichnen können, indem wir eigentlich viele Punkte nebeneinandersetzen. Wir werden noch erfahren, daß MSX-BASIC außerdem noch eine Anzahl sehr starker Anweisungen hat, mit denen wir allerlei Linien zeichnen können.

COLOR Wir können außerdem noch die Farbe des Bildschirmes und die Farben der sogenannten Randgebiete einstellen. Hierbei ist folgende Bildschirmeinteilung zu berücksichtigen:



Die Anweisung, die dies alles bestimmt, hat folgende Form:

`COLOR` Farbvordergrund, Farbhintergrund, Randfarbe

Beispiele:

`COLOR 1`	: nur die Vordergrundfarbe schwarz (1) machen, d.h. zeichnen mit schwarzen Linien
`COLOR ,2`	: Hintergrundfarbe in grün verändern. Beachten Sie das Komma vor der Zahl.
`COLOR 1,2,11`	: Vordergrund schwarz (1), Hintergrund grün(2) und Rand hellgelb(11) machen.
`COLOR ,,11`	: Nur die Farbe des Randes verändern. Beachten Sie, daß vor der Zahl zwei Kommas stehen.

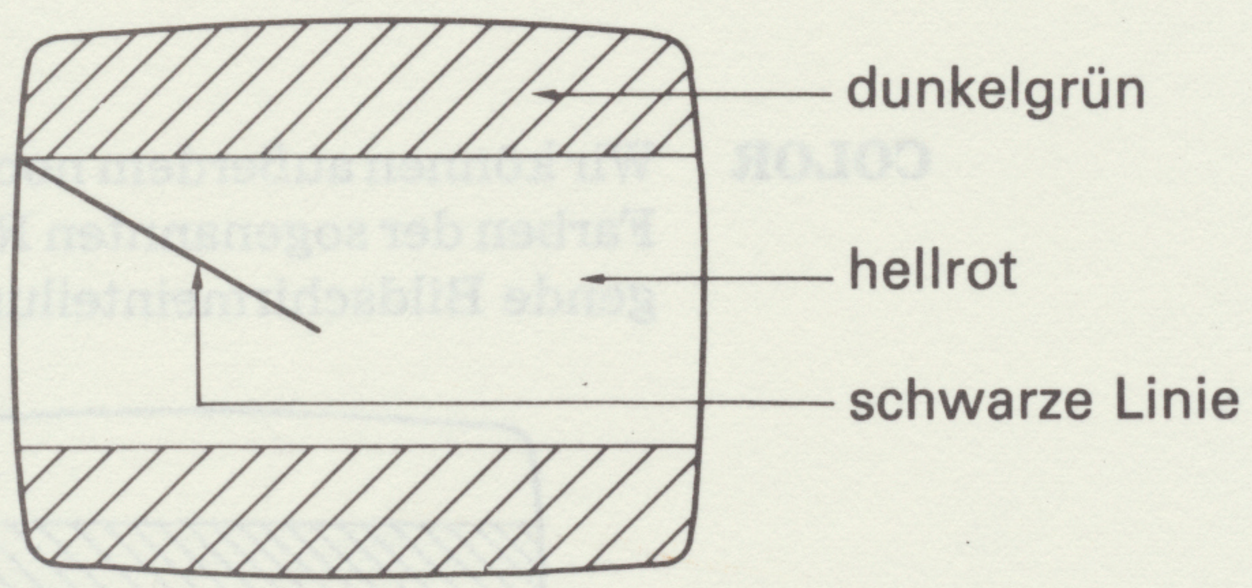
Außerdem ist zu beachten, daß wir immer eine CLS-Anweisung ('clear screen', d.h. Bildschirm löschen) geben müssen, ehe wir die COLOR-Anweisung geben, wenn wir die Hintergrundfarbe verändern wollen. Wenn wir beispielsweise unser voriges Programm erweitern durch:

```
32 CLS
35 COLOR 1,9,12
```

und wir außerdem die Farbkennzeichnung bei der PSET-Anweisung weglassen

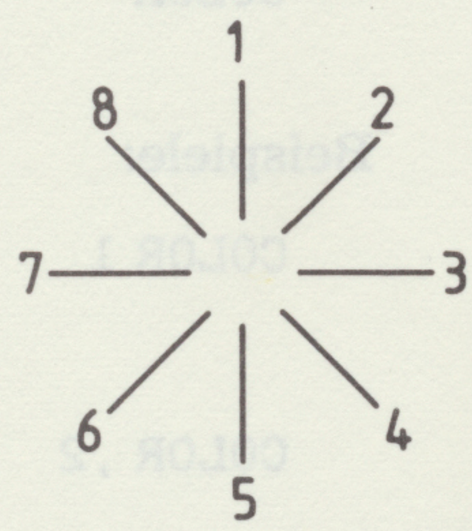
```
50 PSET (J,J)
```

dann erscheint folgendes Bild:



Zeichnen mit dem Joystick

Zum Abschluß dieses Kapitels wollen wir eine hübsche Illustration geben, was wir nun mit der PSET-Anweisung machen können. Wir werden bei diesem Beispiel ein Programm besprechen, mit dem wir eine Zeichnung in hoher Auflösung auf unseren Bildschirm bringen können. Als Zeichenstift benutzen wir dann den Joystick.
Zur Ermittlung der Joystick-Stellung verwenden wir die Funktion STICK (JOYSTICK-Nummer). Diese Funktion ergibt die Werte von 1 bis 8 nach folgendem Schema:



So wird bei der Anweisung D=STICK(1) der Wert 3 zugewiesen, wenn Joystick Nummer 1 nach rechts gezogen wird.
Im Programm wird zugleich von der Möglichkeit Gebrauch gemacht, einen Punkt zu löschen. Dies geschieht dann mit der PRESET-Anweisung:

```
PRESET (X,Y)
```

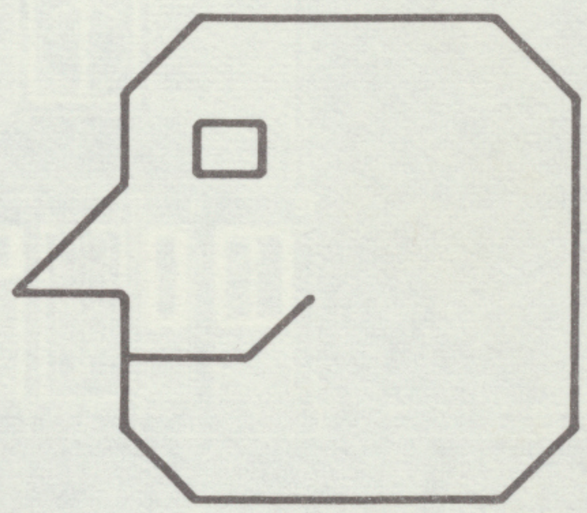
Schließlich wird im Programm ermittelt, ob der Feuerknopf von Joystick 1 eingedrückt wird.
Dies geschieht mit folgender Anweisung:

```
IF STRIG(1) THEN ...
```

Die Anweisung hinter THEN wird hier ausgeführt, wenn der Feuerknopf eingedrückt wird. Nachstehend folgt das vollständige Programm:

```
 10 X=100
 20 Y=100
 30 CLS
 40 SCREEN 2
 50 D=STICK(1)
 60 IF D=1 THEN Y=Y-1
 70 IF D=2 THEN X=X+1:Y=Y-1
 80 IF D=3 THEN X=X+1
 90 IF D=4 THEN X=X+1:Y=Y+1
100 IF D=5 THEN Y=Y+1
110 IF D=6 THEN X=X-1:Y=Y+1
120 IF D=7 THEN X=X-1
130 IF D=8 THEN X=X-1:Y=Y-1
140 IF STRIG(1) THEN GOTO 170
150 PSET (X,Y)
160 GOTO 50
170 PSET (X,Y)
180 FOR K=1 TO 10
190 NEXT K
200 PRESET (X,Y)
210 GOTO 50
```

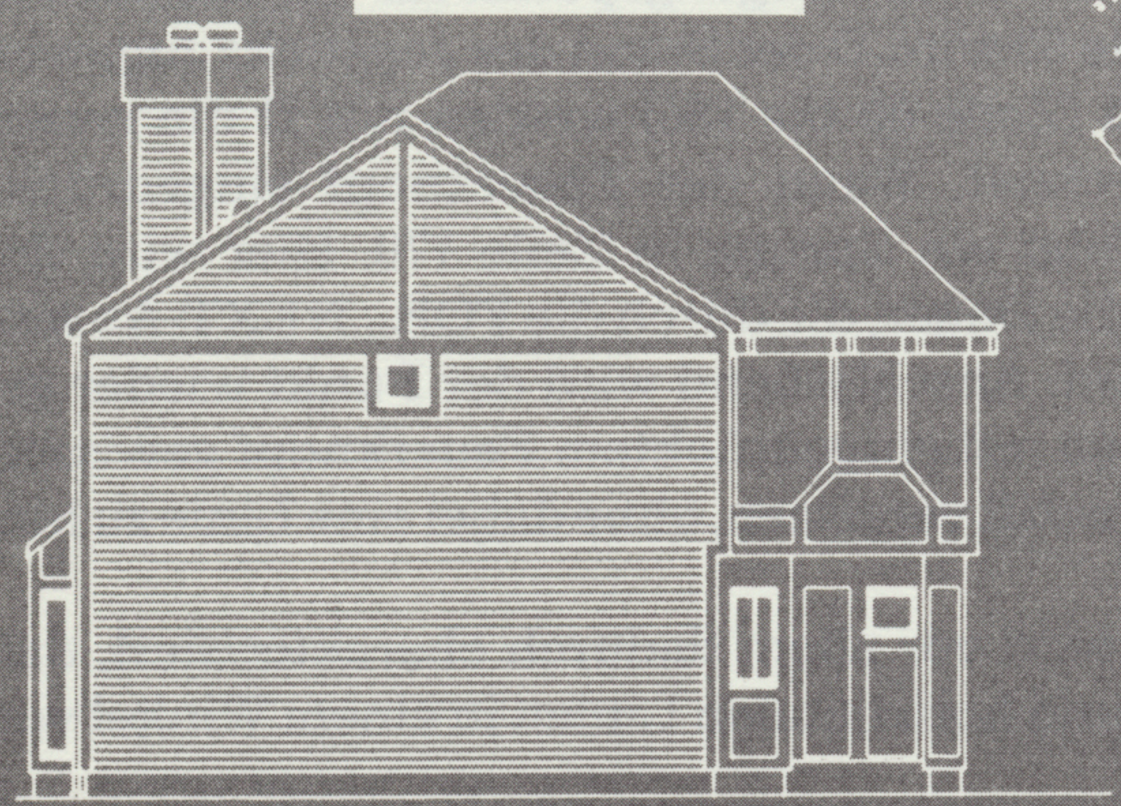
Beispiel eines Resultates:



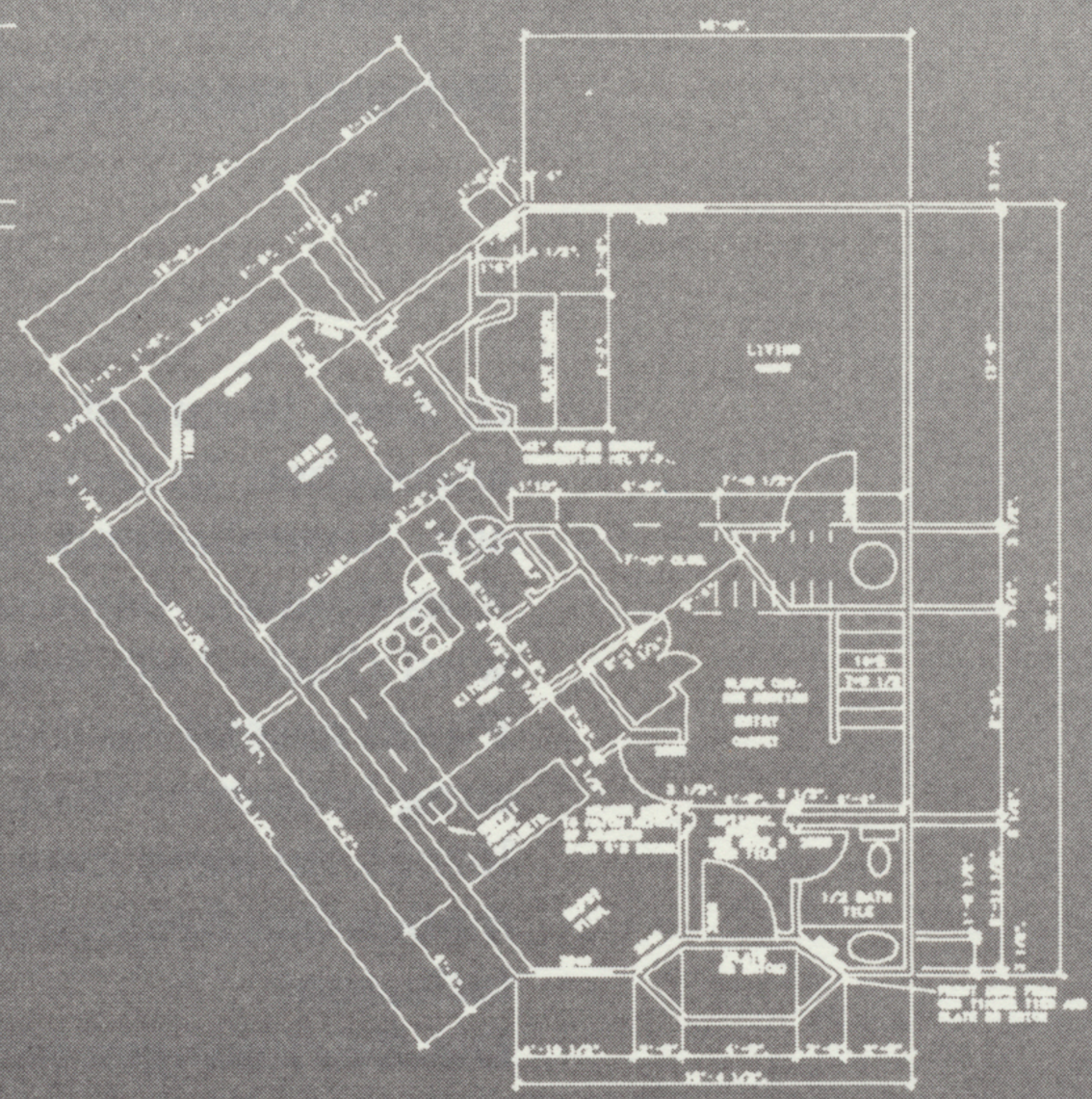
In Zeile 10 und 20 wird ein Ausgangspunkt für die Zeichnung angegeben. In Zeile 50 wird D der Stand des Joysticks zugewiesen. Anschließend (Zeile 60 bis 130) werden mit D die neuen Koordinaten berechnet. Wenn der Feuerknopf eingedrückt wird (Zeile 140), folgt ein Sprung in Zeile 170. Hier wird der Punkt gesetzt für die Dauer, die durch die FOR...NEXT-Anweisung bestimmt wird. Anschließend wird der Punkt wieder gelöscht. So kann man mit eingedrücktem Feuerknopf gewissermaßen den Zeichenstift versetzen, ohne dabei eine Linie zu hinterlassen!
Die Zeichnungen sind vor allem dadurch charakterisiert, daß sie aus vertikalen und horizontalen Linien in einem Winkel von 45° bestehen.

Der Aufbau der Liniendarstellungen verläuft einfach mit Standardanweisungen. Mit der LINE-Anweisung können wir eine Gerade von einem gegebenen Ausgangspunkt zu einem gegebenen Endpunkt ziehen. Die Anweisung DRAW ist ebenfalls auf das Ziehen von geraden Linien ausgerichtet. Hinter DRAW folgt dann eine String, mit der auf einfache Weise angegeben wird, wie eine Verbindung von geraden Linien abzubilden ist. Die CIRCLE-Anweisung bezieht sich auf die Abbildung von Kreisen und Ellipsen. Da wir auch Teile hiervon abbilden können, ist es ganz allgemein eine Anweisung zur Darstellung von Bögen. Die letzte Anweisung PAINT bietet uns die Möglichkeit zur Einfärbung geschlossener Flächen.

VORDERANSICHT



SEITENANSICHT



ERDGESCHOSS

16. Grafische Darstellungen: LINE, DRAW, CIRCLE und PAINT

Einleitung In diesem Kapitel behandeln wir die übrigen Möglichkeiten zur Wiedergabe von Linien auf dem Bildschirm. Mit der PSET-Anweisung können wir zwar bereits alle gewünschten Linien ziehen, doch die nun zu behandelnden Anweisungen erweisen sich in der Praxis als besonders nützlich. Diese Anweisungen gehen davon aus, daß wir in vielen Fällen Figuren aus 'Standardlinien' d.h. Geraden und Bögen, aufbauen können. Die Anweisungen LINE und DRAW bieten die Möglichkeit, auf einfache Weise gerade Linien zu ziehen, während wir mit CIRCLE Bogenlinien (Teile eines Kreises oder einer Ellipse) auf den Schirm bringen können.
All diese Anweisungen gehen von dem grafischen Bildschirm aus, den wir im vorigen Kapitel besprochen haben.

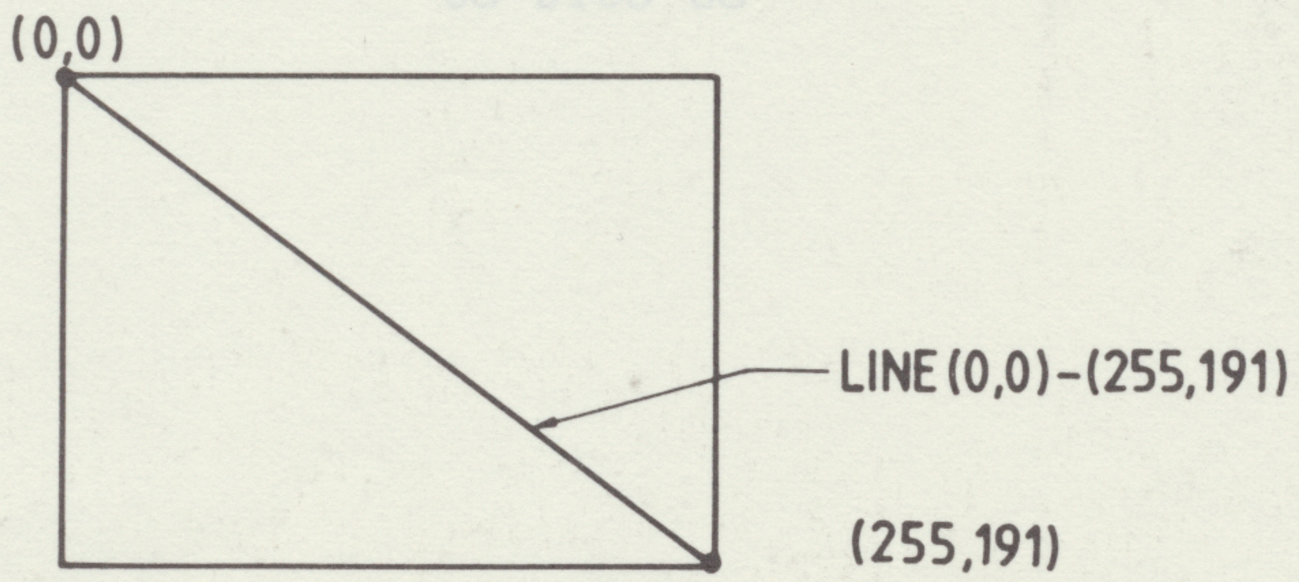
LINE Die einfachste Form der LINE-Anweisung ist:

```
LINE (X1,Y1)-(X2,Y2)
```

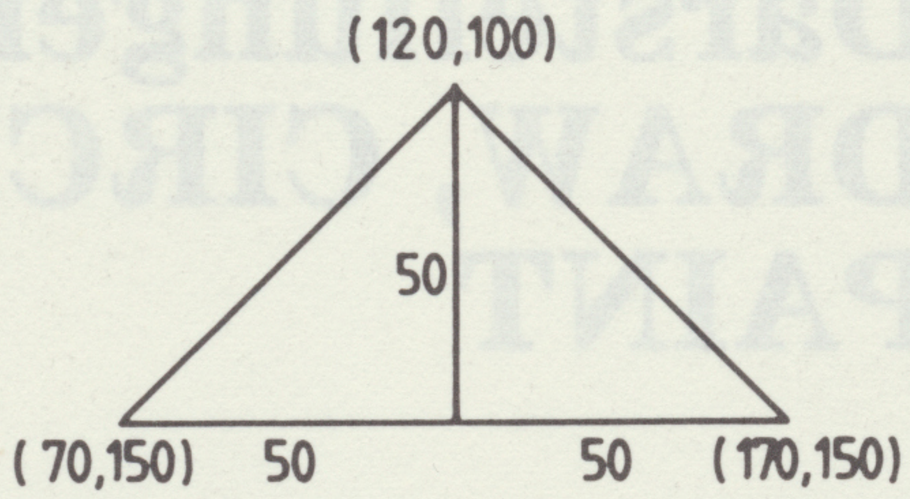
Mit (X1,Y1) wird der Anfangspunkt der Linie angegeben, mit (X2,Y2) der Endpunkt.
Beispiel:

```
10 SCREEN 2
20 LINE (0,0)-(255,191)
30 GOTO 30
```

Als Resultat sehen wir nun eine schräge Linie auf dem Bildschirm.



Im nachstehenden Programm werden wir gemäß der unten angegebenen Definition ein Dreieck zeichnen:



Programm:

```
10 SCREEN 2
20 LINE ( 70,150)-(170,150)
30 LINE (170,150)-(120,100)
40 LINE (120,100)-( 70,150)
50 GOTO 50
```

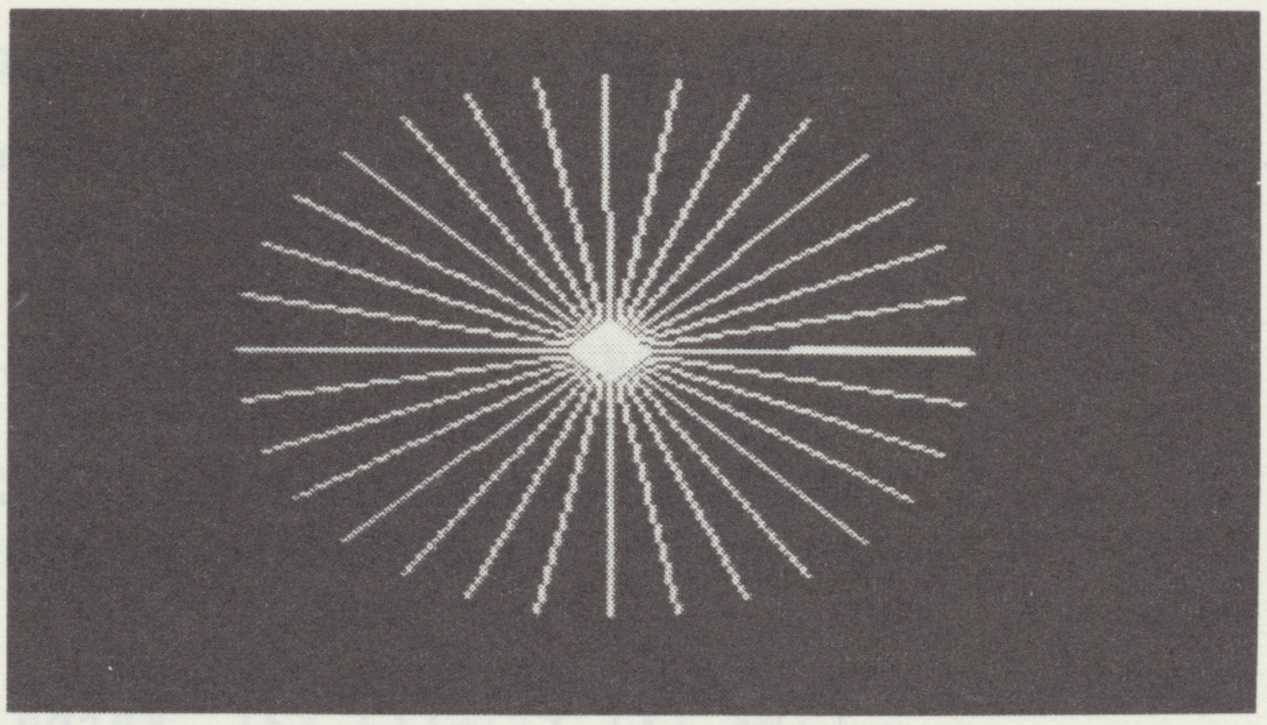
Wir können eventuell die Anfangsposition weglassen. In diesem Fall wird der Endpunkt als Ausgangspunkt genommen, der auf Grund einer vorigen Anweisung erreicht wurde. So erhalten wir das gleiche Dreieck mit folgendem Programm:

```
10 SCREEN 2
20 LINE  ( 70,150)-(170,150)
30 LINE -(120,100)
40 LINE -( 70,150)
50 GOTO 50
```

Im folgenden Programm werden wir die LINE-Anweisung zur Darstellung eines hübschen Sternes anwenden.
Programm:

```
10 SCREEN 2
20 AL=2*3.14159
30 FOR K=0 TO AL STEP AL/32
40 X=INT (70*COS(K))+122
50 Y=INT (70*SIN(K))+95
60 LINE (122,95)-(X,Y)
70 NEXT K
80 GOTO 80
```

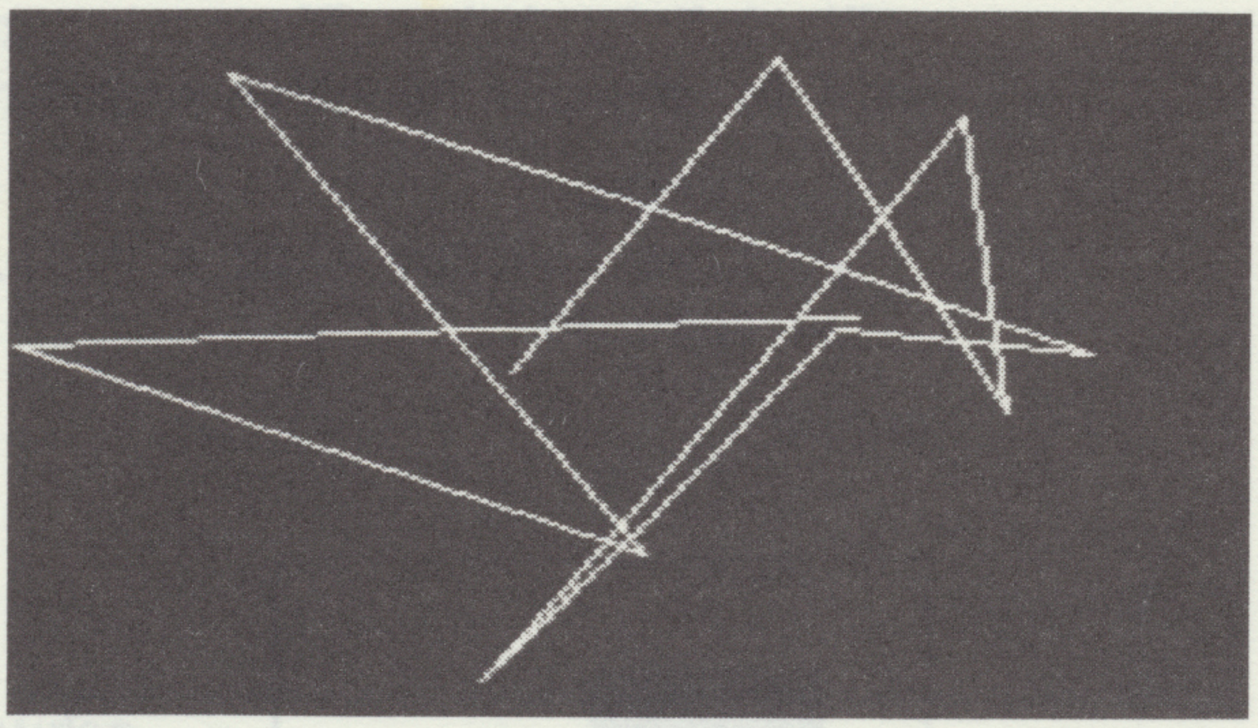
Resultat:



Als letztes Beispiel behandeln wir ein Programm, mit dem wir ein Wirrwarr von Linien auf den Bildschirm bekommen.

```
10 SCREEN 2
20 PSET (100,100)
30 FOR K=1 TO 10
40 X=INT(RND(1)*255)
50 Y=INT(RND(1)*191)
60 LINE -(X,Y)
70 NEXT K
80 GOTO 80
```

Resultat:



Nimmt man statt FOR K=1 TO 10 beispielsweise FOR K=1 TO 100, so wird das Muster wirklich sehr launisch.

Wir können einer Linie außerdem eine Farbe geben, und dies geschieht auf gleiche Weise, wie bei PSET, wie gesagt durch ein Kommazeichen und eine Farbnummer, z.B.

```
LINE (X1,Y1)-(X2,Y2), Farbkode
```

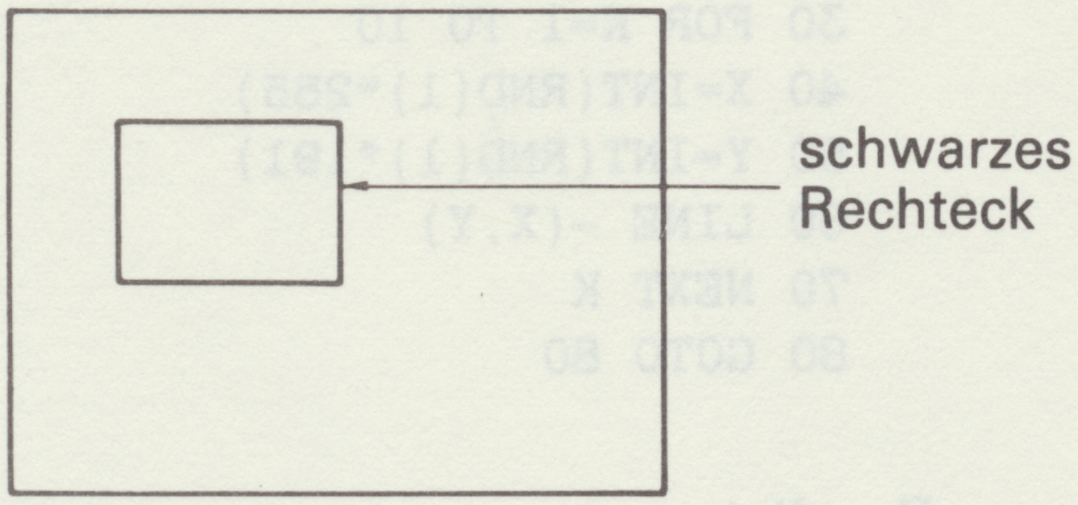
Hierbei ist jedoch zu berücksichtigen, daß am Schnittpunkt von zwei Linien mit verschiedenen Farben Unregelmäßigkeiten auftreten können. Die gekreuzte Linie kann zu einem kleinen Teil rings um den Schnittpunkt (höchstens 8 Punkte) die Farbe der anderen Linie annehmen.
Schließlich können wir mit der LINE-Anweisung leicht Blöcke zeichnen. Zu diesem Zweck erweitern wir die Anweisung mit ,B:

```
LINE (X1,Y1)-(X2,Y2), Farbe ,B
```

Beispiel:

```
10 SCREEN 2
20 LINE (50,50)-(100,100),1,B
30 GOTO 30
```

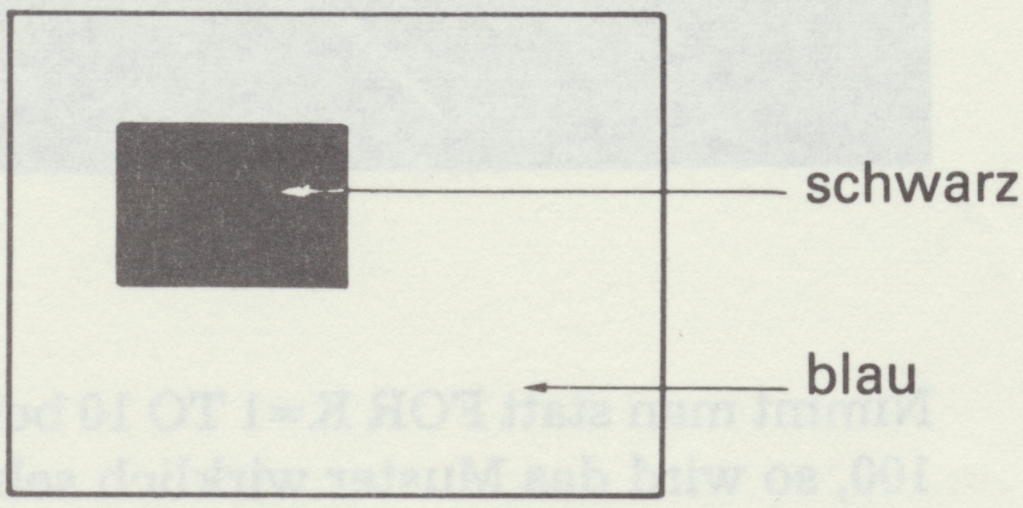
führt zu dem Resultat:



Beachten Sie, daß wir den Block durch LINE-Anweisung von zwei einander schräg gegenüberliegenden Winkeln offenbar völlig definiert haben. Wenn wir nun statt B ein BF eingeben, wird das gesamte Rechteck eingefärbt, z.B.:

```
10 SCREEN 2
20 LINE (50,50)-(100,100),1,BF)
30 GOTO 30
```

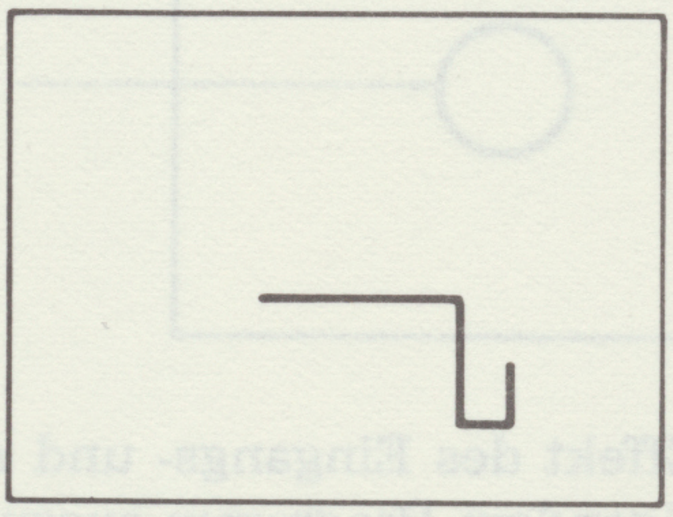
ergibt:



DRAW Die DRAW-Anweisung ist eine besonders kräftige Anweisung zum Zeichnen waagerechter, senkrechter und schräger Linien. Hier wieder ein einfaches Beispiel:

```
10 SCREEN 2
20 PSET (127,95)
30 DRAW "R50D50R25U25"
40 GOTO 40
```

Resultat:



Zeile 20 gibt den Ausgangspunkt unserer Zeichenübung an. Die anschließende DRAW-Anweisung gibt mit einer String an, was gezeichnet werden soll:

R50 Zeichne eine Linie 50 Punkte nach rechts.

D50 Nun 50 Punkte nach unten gehen (D ist abgeleitet von 'down', d.h. 'nach unten).

R25 Anschließend wieder 25 Punkte nach rechts.

U25 Nun wieder 25 Punkte nach oben (U ist abgeleitet von 'up', d.h. 'nach oben').

Eine vollständige Übersicht der DRAW-Möglichkeiten ist im Anhang D zu finden.

CIRCLE Dies ist die letzte Anweisung zum Zeichnen von Linien, und in diesem Fall von Kurven. Die allgemeine Form ist:

`CIRCLE (X,Y) ,` Radius, Farbe, Ausgangswinkel, Endwinkel, Perspektive.

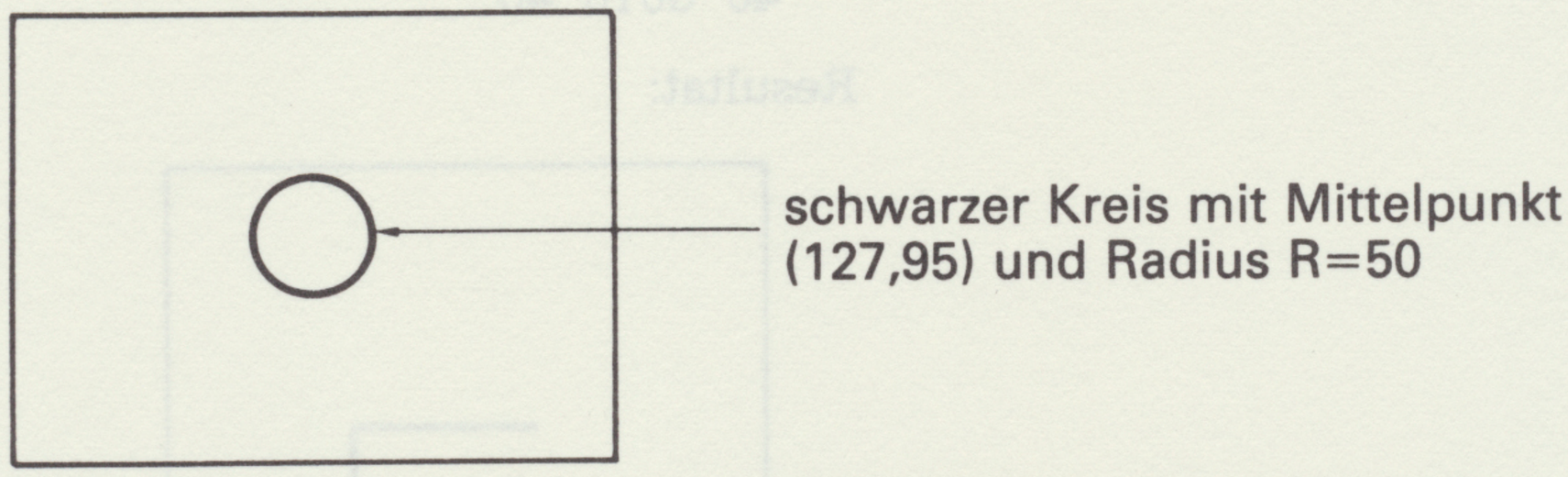
mit:

(X,Y)	Koordinaten des Mittelpunktes
Radius	Wert des Radius
Farbe	Farbkode
Ausgangs-winkel	Winkel, von dem aus die Kurve gezeichnet wird (Angeben im Bogenmaß: 0..2 π)
Endwinkel	Winkel zum Endpunkt der Kurve (Angeben im Bogenmaß: 0..2 π)
Perspektive	Zahlenangabe darüber, inwiefern der Kreis abgeplattet, d.h. als Ellipse gezeichnet wird. Trägt man hier den Wert 1.4 ein, so erhält man einen Kreis.

Wir geben hier einige Beispiele:

```
10 SCREEN 2
20 CIRCLE (127,95),50,1,,,1.4
30 GOTO 30
```

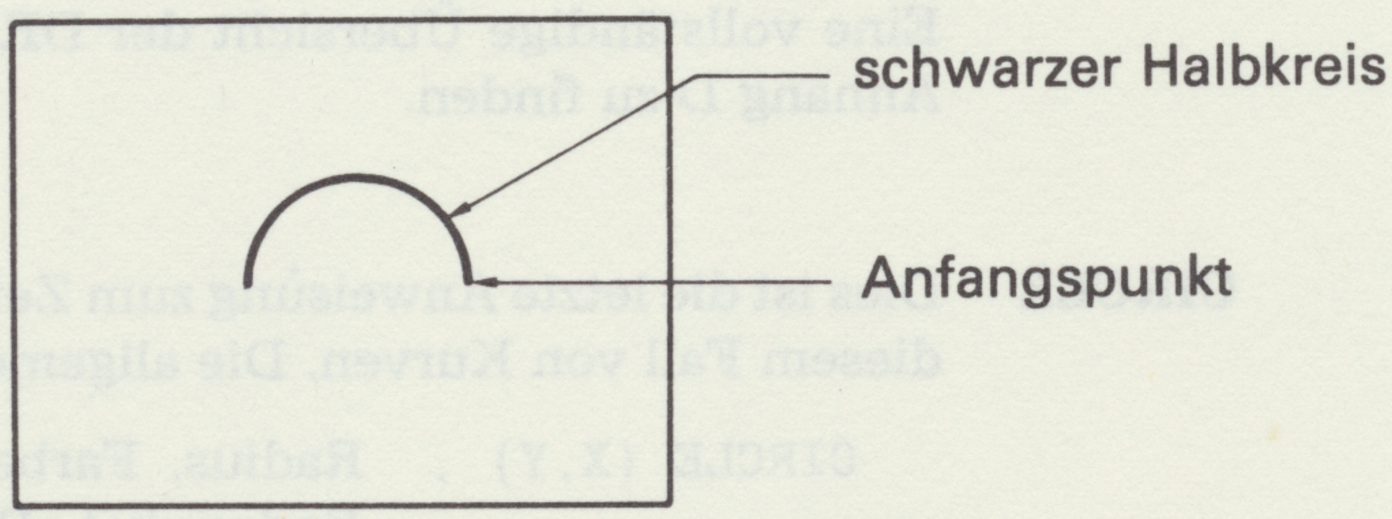
Resultat:



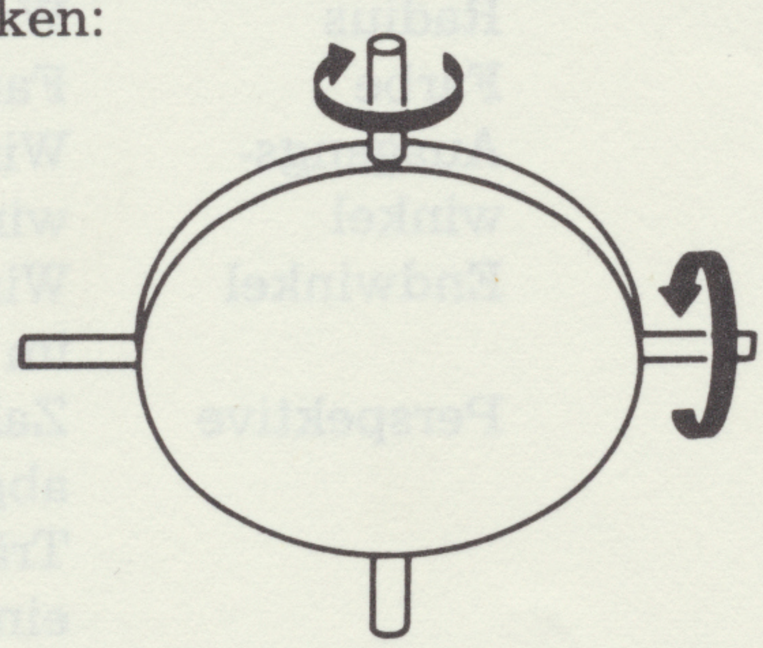
Den Effekt des Eingangs- und Ausgangswinkels können wir mit folgendem Programm aussuchen:

```
10 INPUT B
20 INPUT E
30 SCREEN 2
40 CIRCLE (127,95),50,1,B,E,1.4
50 GOTO 50
```

Setzt man nun für B den Wert 0 und für E den Wert 3.1415 ein, so erhält man folgenden Halbkreis:



Die letzte Zahl der CIRCLE-Anweisung enthält das sogenannte perspektivische Verhältnis. Wir können hier am besten an folgendes Bild denken:



Wird die Scheibe um eine der Achsen gedreht, so sehen wir sie unter einem ständig wechselnden Winkel, d.h. perspektivischen Verhältnis.
Dies führt dazu, daß wir eine Ellipse in folgenden Formen sehen:

und

Achsendrehung Vertikal

Achsendrehung horizontal

Wir können diesen Effekt am besten mit folgendem Programm untersuchen:

```
10 INPUT K
20 SCREEN 2
30 CIRCLE (127,95),50,1,,,K
40 GOTO 40
```

PAINT Die letzte Anweisung, die wir in diesem Kapitel besprechen, ist die PAINT-Anweisung. Hiermit können wir bestimmte Flächen einfärben, indem wir einen beliebigen Punkt in solch einer Fläche anweisen.
Die allgemeine Form lautet:

```
PAINT (X,Y), Farbfläche, Farblinie
```

mit

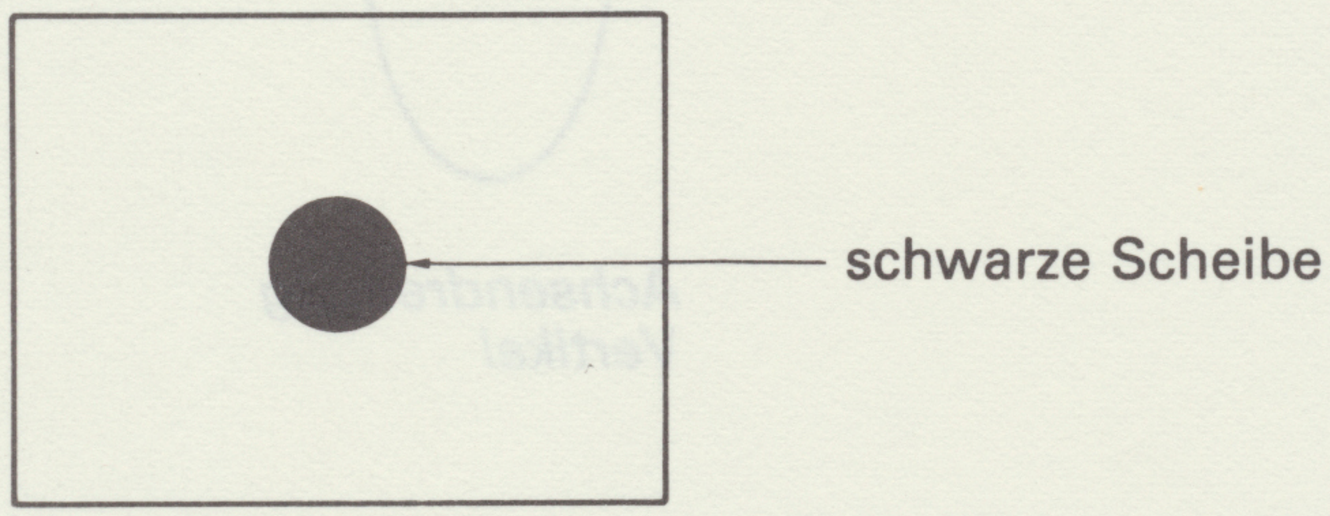
(X,Y)	Koordinaten des anzuweisenden Punktes
Farbfläche	Farbe, in der die Fläche einzufärben ist
Farblinie	Farbe, in der die Flächengrenze angegeben werden muß

Üblicherweise lassen wir den letzte Kode weg, da für den grafischen Schirm gilt, daß die Farbe der Grenzlinie gleich der Flächenfarbe sein muß. Da wir immer die Grenzen der Fläche angeben müssen, bevor wir sie färben, liegt die Farbenkodierung praktisch bereits fest.

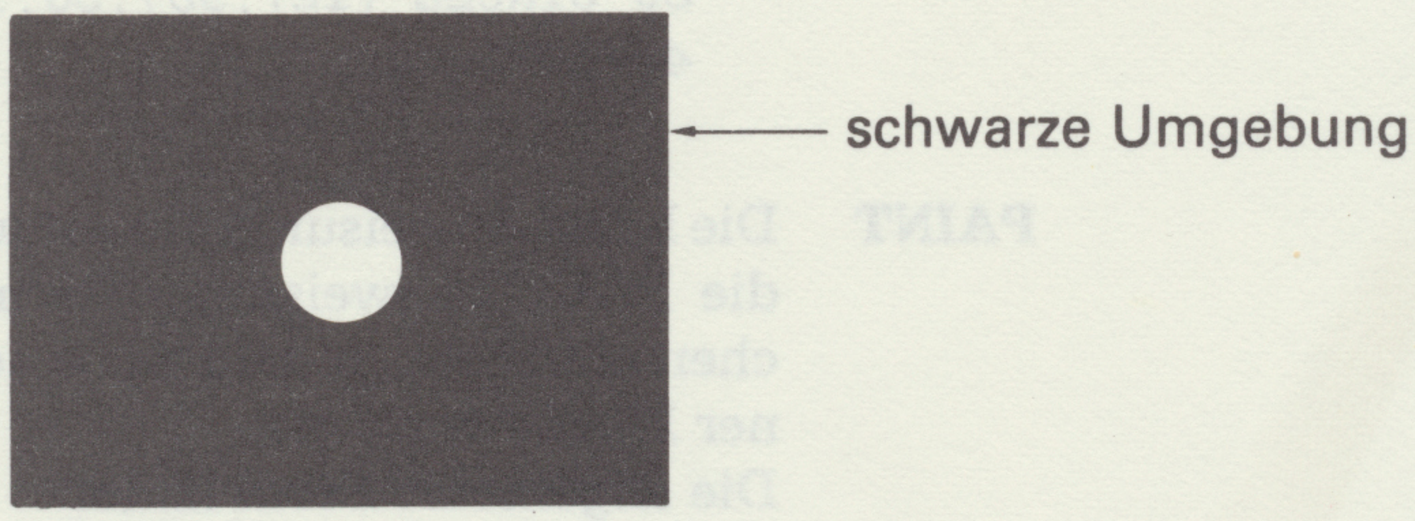
Ein Beispiel:

```
10 INPUT X,Y
20 SCREEN 2
30 CIRCLE (127,95),50,1,,,1.4
40 PAINT (X,Y),1
50 GOTO 50
```

Wenn wir nun mit PAINT einen Punkt innerhalb des Kreises anweisen, dann erhalten wir folgendes Ergebnis:



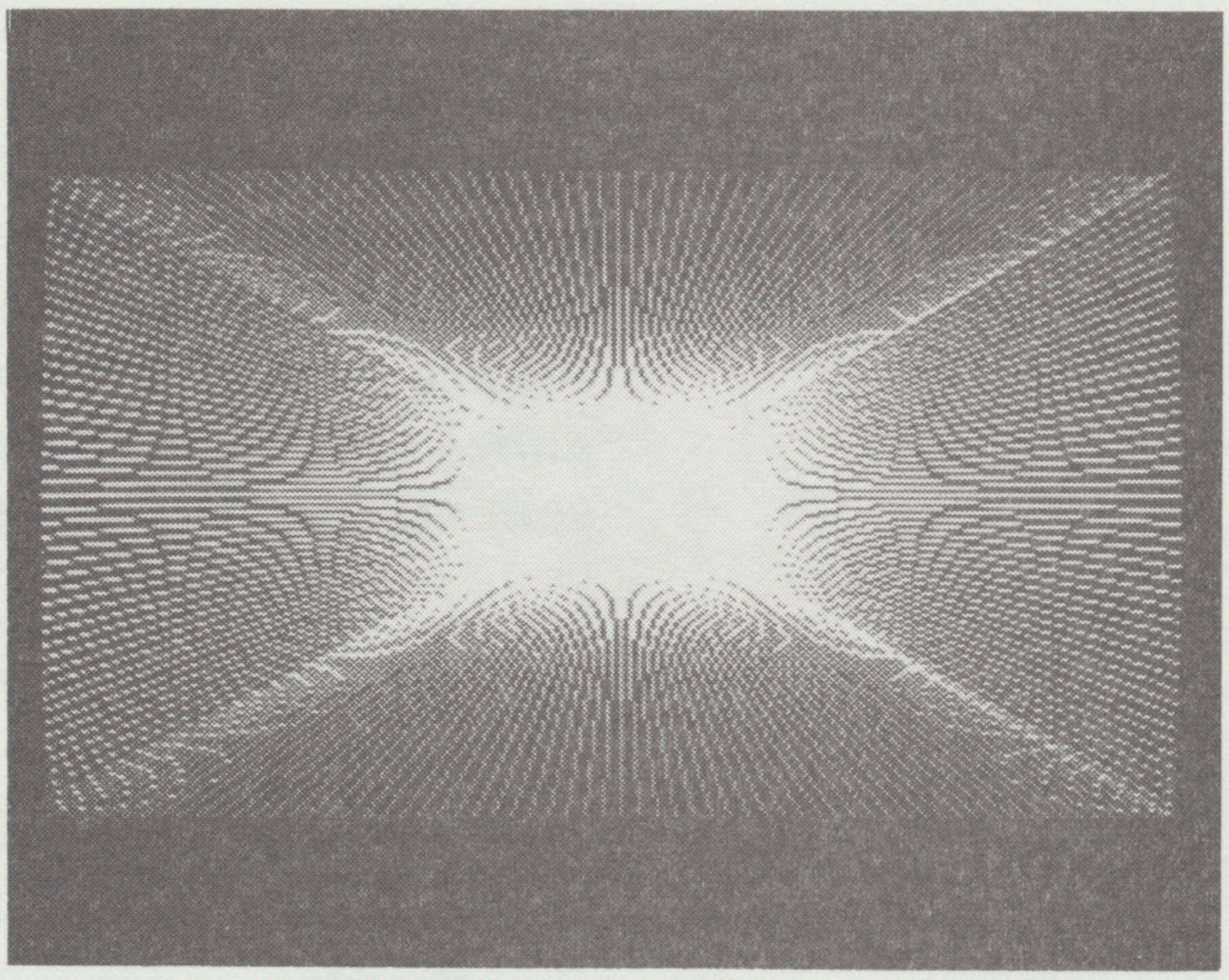
Wir sehen, wie nun die Fläche innerhalb des Kreises eingefärbt wird. Weisen wir jedoch einen Punkt außerhalb des Kreises an, so erhalten wir:



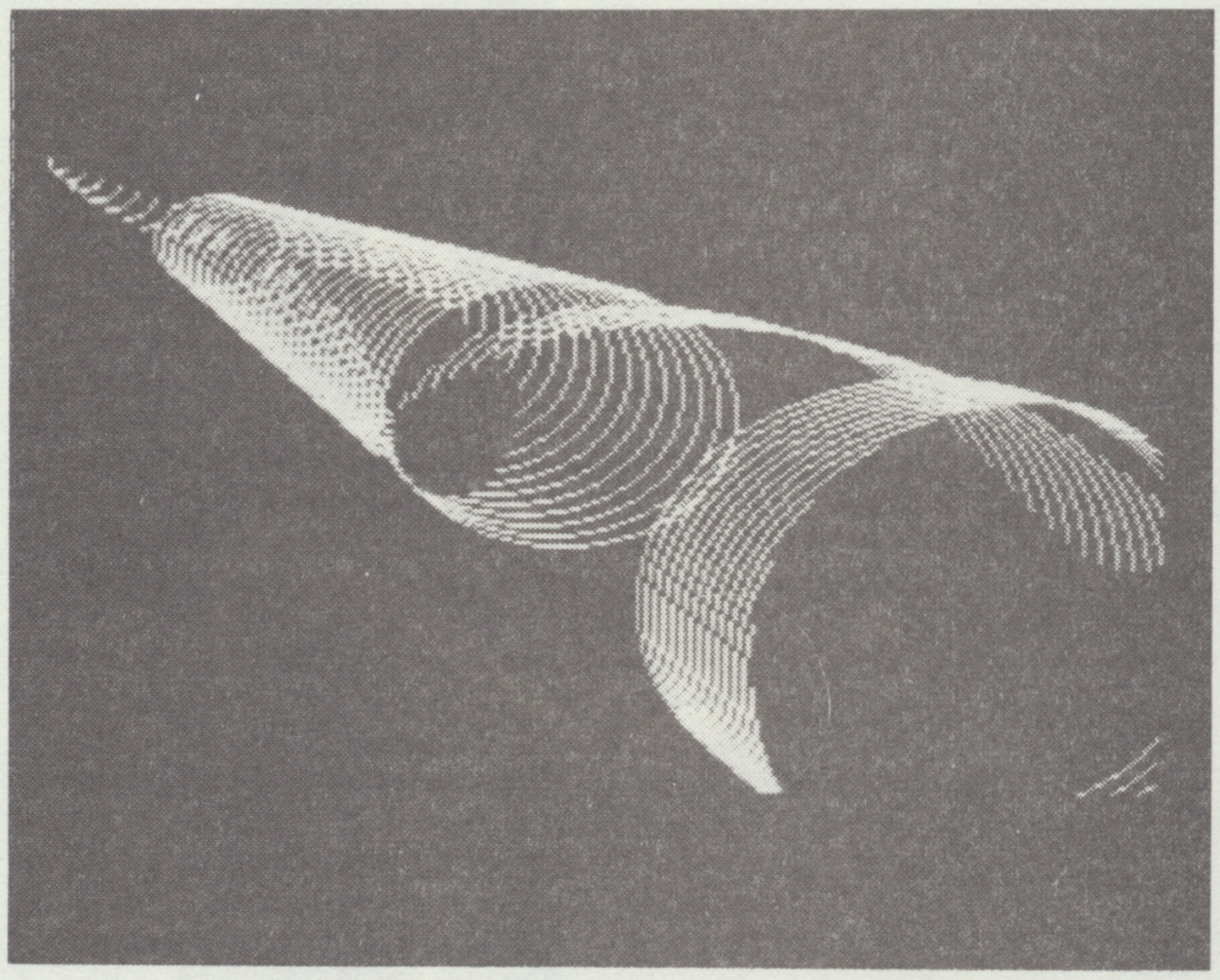
In diesem Fall wird also das andere Gebiet gefärbt.

Beispiele

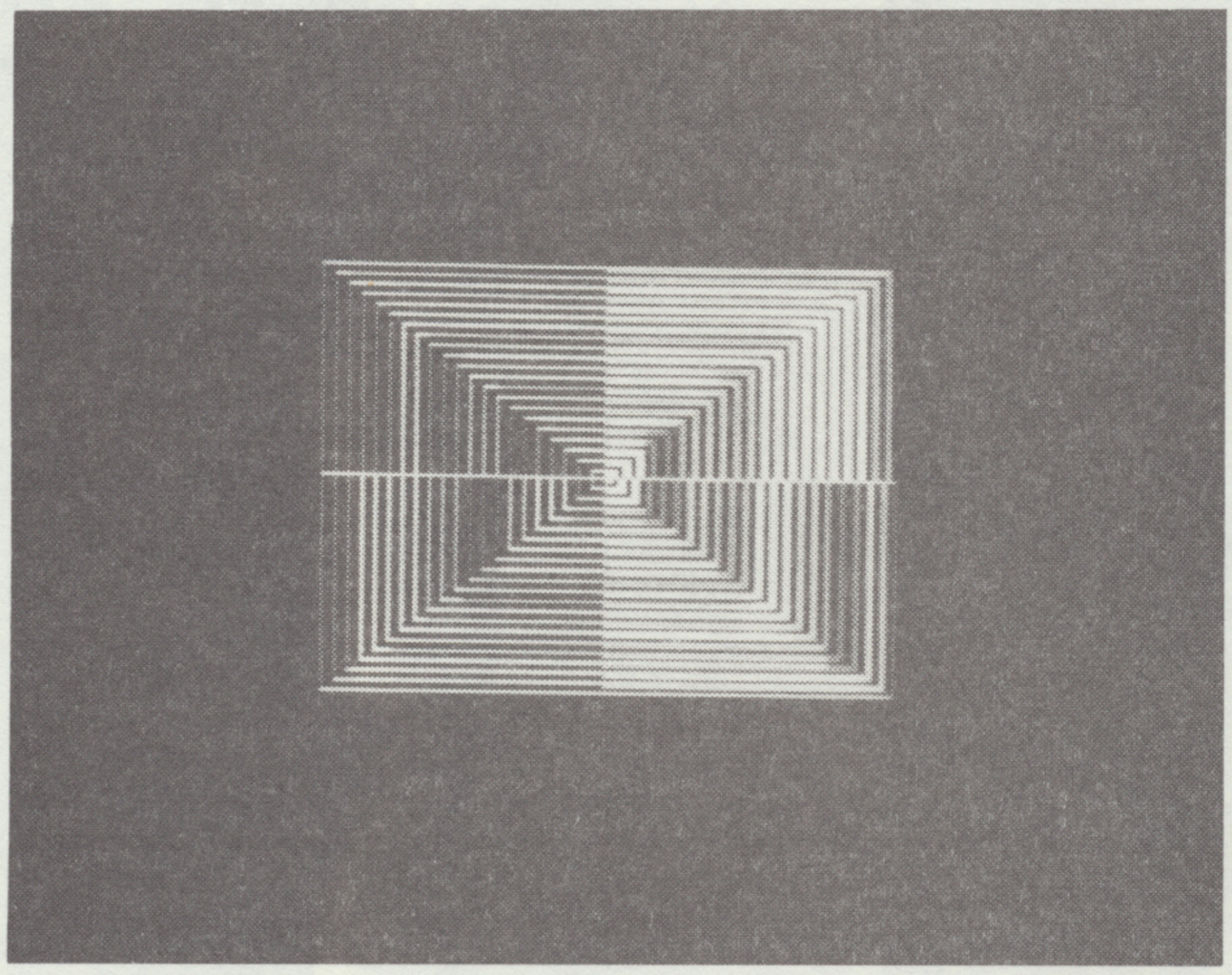
```
10 REM STERN AUS LINIEN
20 SCREEN 2
30 FOR T=0 TO 256 STEP 4
40 LINE (T,0)-(256-T,192)
50 LINE (0,T)-(255,192-T)
60 NEXT T
70 GOTO 70
```



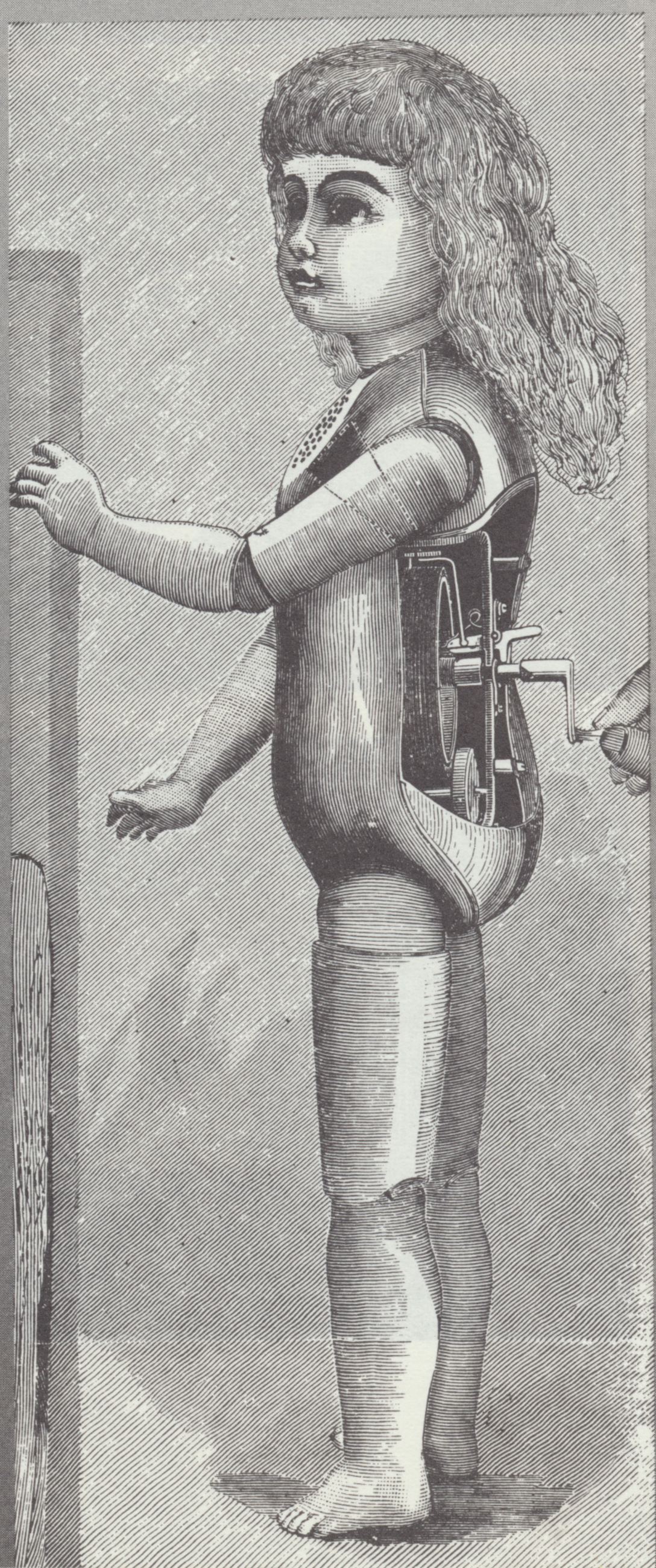
```
10 REM TUNNEL
20 PI=3.14159
30 SCREEN 2
40 FOR R=1 TO 80
50 CIRCLE(3*R,2*R),R,15,PI+SIN(R/
   40*PI)*PI,PI+COS(R/80*PI)*PI
60 NEXT R
70 GOTO 70
```



```
10 REM RECHTECKE
20 SCREEN 2
30 DRAW "BM128,96"
40 FOR I=4 TO 255 STEP 3
50 XC=I MOD 13+2
60 XA=(I/4) MOD 3
70 DRAW"S=I;C=XC;A=XA;URDL2UR2DL"
80 NEXT I
90 GOTO 90
```



Dieses Kapitel ist wieder ein Intermezzo, d.h. es werden keine neuen Anweisungen behandelt. Zum Schluß der Behandlung von BASIC ist ein Rückblick angemessen. Wir gehen dabei tiefer auf die Entwicklung von Automaten ein. Hierbei werden nacheinander die Zeiträume besprochen, in denen der Mensch Methoden zur Wiedergabe von Zahlen entdeckte, Automaten erfand und über sein eigenes Denken nachdachte.

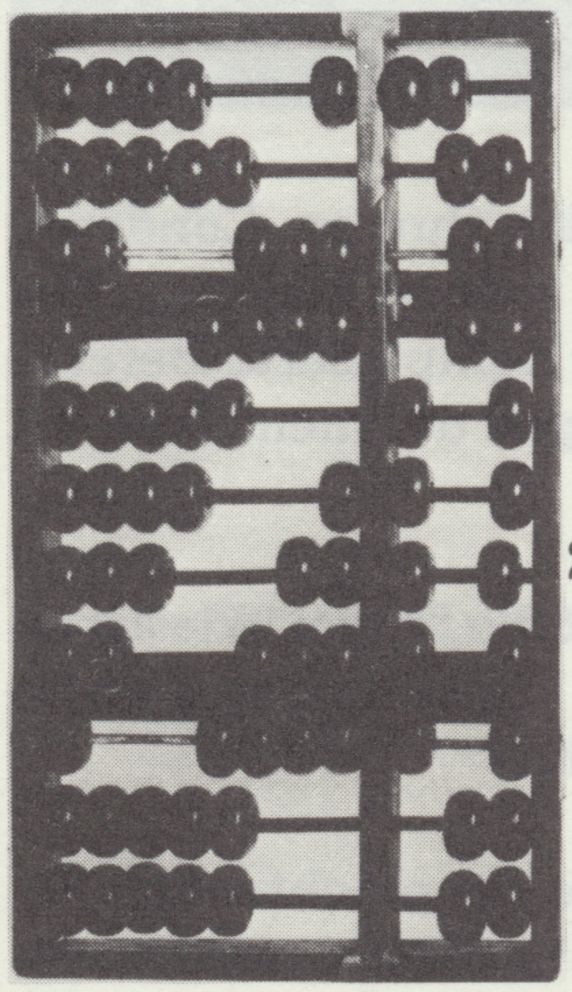


17. Intermezzo: über einst und jetzt

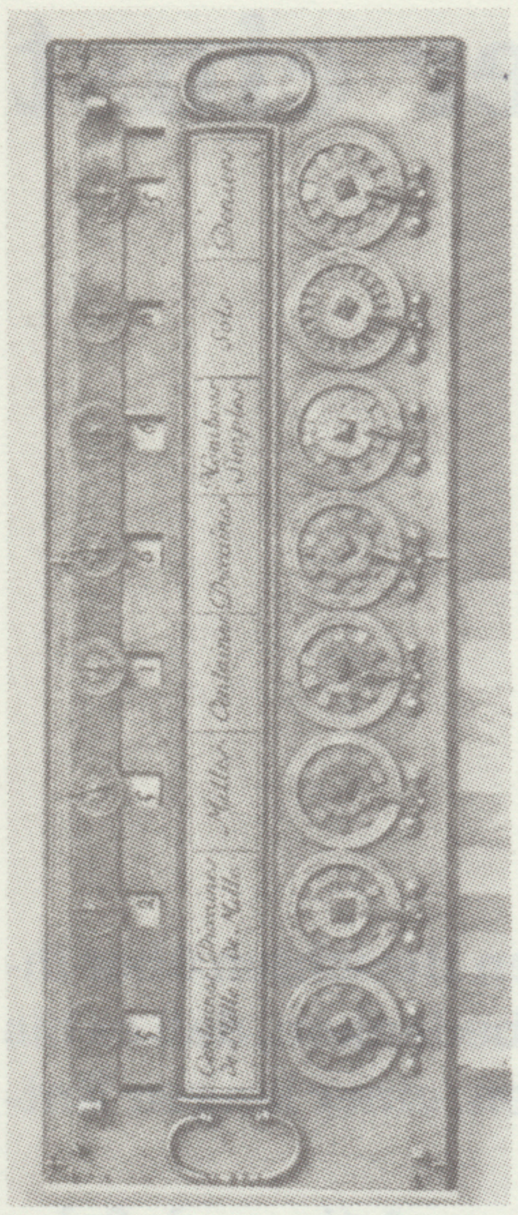
Einleitung Computer sind komplizierte Apparate. Allen Berichten über ihre Einfachheit zum Trotz müssen wir doch bedenken, daß es bis zu dem Computer, wie wir ihn heute kennen, einer langen Entwicklungszeit bedurfte.
Es ist durchaus interessant, sich zu vergegenwärtigen, daß der Mensch fortwährend auf der Suche war Vorgänge zu automatisieren. Automatisierung bedeutet: Ablauf einer ganzen Reihe von Handlungen, ohne daß sich der Mensch einschalten muß. Es gibt prachtvolle Beispiele sehr raffiniert durchdachter Apparate.

Entwicklungen Die Entwicklungen, die die geschichte uns lehrt, können eigentlich auf drei Punkte zurückgeführt werden:

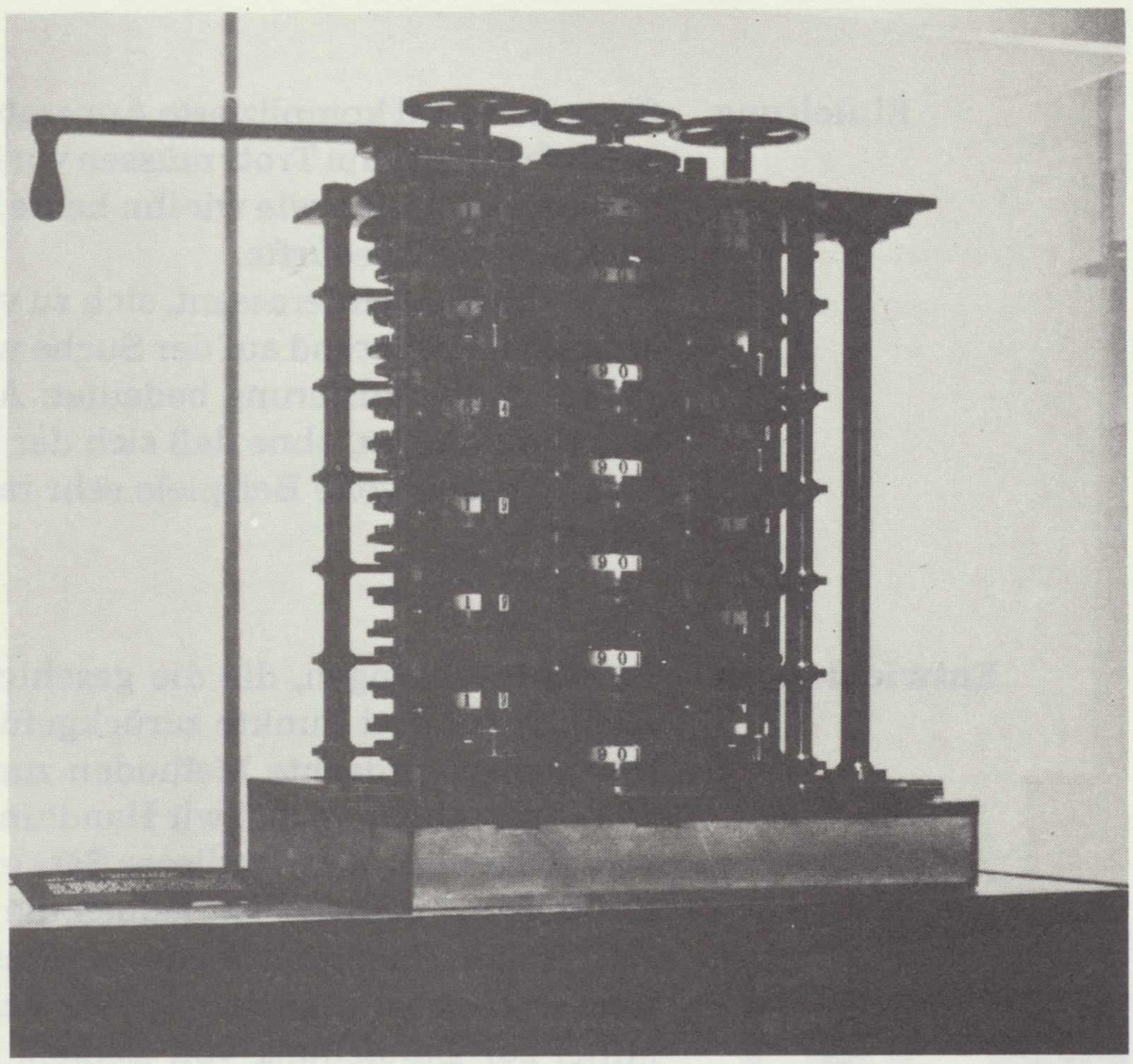
1 Der Mensch erdachte Methoden zur Festlegung von Größen, wie etwa Zahlen, so daß wir Handlungen damit ausführen können. Wahrscheinlich ist dieser Satz etwas schwer zu begreifen. Ein Beispiel verdeutlicht jedoch die Situation.
Auf nebenstehender Abbildung sehen wir ein Rechenbrett. Ein Rechenbrett ist praktisch nichts anderes als ein praktisches Mittel zur Darstellung von Zahlen. Es ist praktisch, weil wir hiermit allerlei mathematische Berechnungen ausführen können. Die eigentlichen Berechnungen mit solch einem Rechenbrett werden natürlich vom Menschen ausgeführt: er schiebt die Kugeln hin und her. Hier läßt sich natürlich noch überhaupt kein Automat erkennen.



2 Der Mensch erdachte Mechanismen zur automatischen mathematischen Berechnung. Ein interessantes Beispiel ist hier die Rechenmaschine des berühmten französischen Mathematikers und Philosophen Blaise Pascal aus dem Jahre 1642. Wieder handelt es sich um eine Methode zur praktischen Darstellung von Zahlen, diesmal mit Rädchen. Die Stellung eines Rädchens gab eine Zahl an. An sich ist dies noch kein Unterschied zu dem zu-



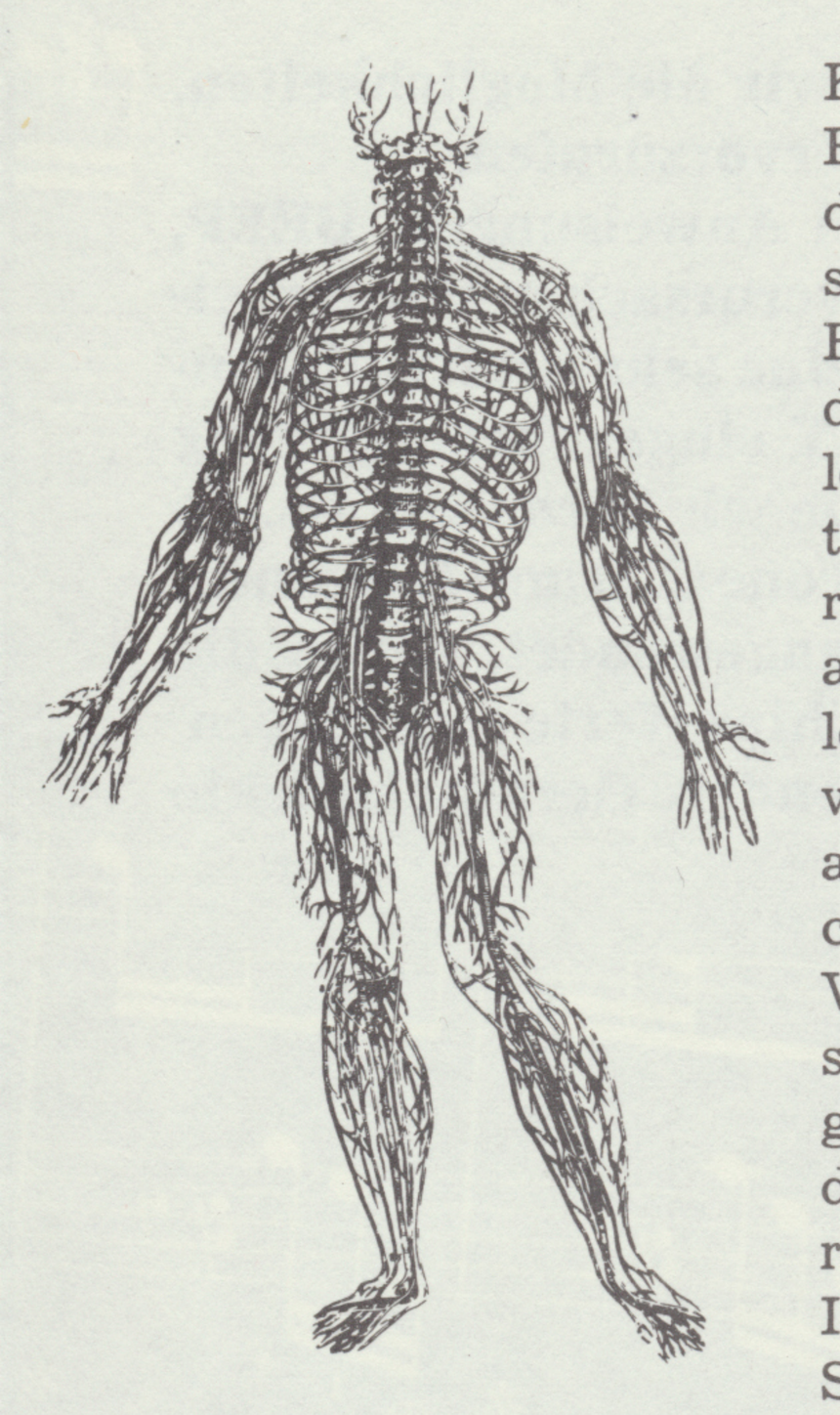
vor genannten Rechenbrett, denn auch da handelte es sich um eine praktische Art der Darstellung von Zahlen. Es gibt jedoch durchaus einen Unterschied, denn bei Pascal sind die Rädchen durch einen derartigen Mechanismus miteinander verbunden, daß wir hiermit Berechnungen ausführen können. Wir erkennen also, daß neben einer praktischen Form der Darstellung von Zahlen ein Mechanismus eingeführt wird, mit dem wir durchaus Bearbeitungen vornehmen können.
Spätere Entwicklungen, wie etwa die 'Analytic Engine' von Charles Babbage (1791-1871), benutzten dies immer als Ausgangspunkt.



Die 'Analytic Engine' kann man im berühmten Londoner Science Museum bewundern. Sie ist ein äußerst kompliziertes mechanisches Gerät, mit dem man sehr komplizierte Berechnungen ausführen kann: ein Musterbeispiel der menschlichen Vernunft und Sachkunde.

3 Der Mensch besann sich immer mehr auf die Frage: 'Wie arbeiten wir Menschen eigentlich?' Ich habe absichtlich den etwas primitiven Ausdruck 'arbeiten' gebraucht.

Nebenstehende Abbildung zeigt uns eine der prachtvollen Radierungen, die uns Andreas Vesalius (1514-1564), Leibarzt von



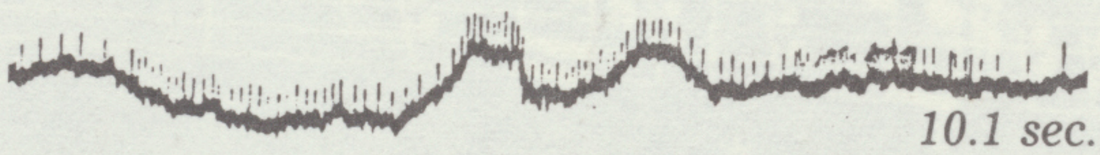
Karl V., hinterließ. Sein anatomisches Werk ist eine logische Fortsetzung von Leonardo da Vinci, der 1519 starb. Die Abbildung zeigt uns das Nervensystem des Menschen außerhalb dessen, was wir nun für das Wesentlichste halten: das menschliche Hirn. Der Einblick in das Nervenzentrum beschränkte sich lediglich darauf, daß es sich um Bahnen handelte, über die Befehle geschickt wurden, beispielsweise um den Muskeln den Auftrag zu geben, sich zu spannen. Man war sich noch nicht im klaren über die Bedeutung des Gehirnes (und in gewisser Hinsicht auch des Rückenmarks) als Zentrum, von dem aus diese Befehle - die Information - koordiniert wurden. Diese Einsicht gewann man erst Mitte des 19. Jahrhunderts (P. Broca, 1824-1887), als man entdeckte, daß gewisse Mängel im Hirn zu sprachlichem Unvermögen führen.

Von diesem Augenblick an ist das Denken über unser Nervensystem und das Denken über unser Denken in eine Beschleunigung geraten. Das Nachdenken über unser Nervensystem wurde mehr und mehr zum Nährboden für das Entwerfen neuer Geräte.

In diesem Prozeß ist die Entdeckung sehr wesentlich, daß die Signale, die wir den Nerven entnehmen können, immer eine sehr einfache Form haben.

Das nachstehende Foto zeigt die Aufzeichnung von Signalen, die aus den Nervenbahnen vom Gehirn aus gemessen wurden. Am auffälligsten ist es, daß die Signale aus kurz andauernden Pulsen, den 'spikes' bestehen. Wenn man dies erkennt, kommt man schon rasch zu der Vorstellung, daß die Information in unserem Hirn mit diesen Pulsströmen verbunden ist. Außerdem ergibt sich die Vorstellung, daß unsere Hirnzellen offenbar mit Schaltungen ausgestattet sind, durch die Handlungen in derartigen Pulsströmen ausgeführt werden.

Quelle: K.L. Boon: Elektrofysiologische lokalisatie, Bericht Bio-informatica THT,1968



Wüßten wir überhaupt nichts von Computern und betrachteten dann eines dieser elektrischen Signale, die durch unsere Nerven (elektrische Leitungsdrähte) des Computers laufen, dann würden wir auch derartige Signale aufnehmen. Mit dieser Darlegung wollen wir nun keineswegs behaupten, daß unser Computer genau wie unser Hirn arbeitet. Wir wollen allerdings sagen, daß wir nun an dem Punkt angelangt sind, in dem Größen (Zahlen, Buchstaben usw.) offenbar auf ganz allgemeine Weise dargestellt werden und es offenbar auch sehr allgemeine Schaltungen zur Verarbeitung dieser Signale gibt.

In diesem Zusammenhang dürfen wir nicht vergessen, daß einer der Pioniere des modernen Computers, John v. Neumann, durch dieses Thema so fasziniert war, daß er sogar ein Buch darüber schrieb (Das Nervensystem als Computer).

In diesem Kapitel besprechen wir die Möglichkeiten, mit unserem Computer Töne hervorzurufen.
MSX-BASIC bietet hierfür drei Anweisungen. BEEP, PLAY und SOUND. Mit BEEP verursachen wir nur einen kurzen Piepton. PLAY ist eine sehr kraftvolle Anweisung. Mit einer hinter PLAY eingegebenen String bestimmen wir eine Melodie, die wir hören wollen. So läßt "CDE" die Reihenfolge der Töne C, D und E ertönen. Die SOUND-Anweisung wird angewendet, um in die Register des gesonderten Ton-Chips Werte einzugeben. Hiermit können wir allerlei besondere Geräuscheffekte erzielen.

18. Ton: BEEP, PLAY und SOUND

Einführung Geräusch und Klang spielt in unserem Leben eine wichtige Rolle, und daher ist es nicht verwunderlich, daß unser Computer uns auch viele Möglichkeiten in diesem Bereich bietet. Besser gesagt, unser Computer bietet hier viel umfangreichere Möglichkeiten, als die großen Computer, die im Wirtschaftsleben und in großen akademischen Rechenzentren gebraucht werden. Der Grund für diesen auffälligen Unterschied ist, daß wir unseren Computer u.a. für Schulungsprogramme und Spiele verwenden wollen, und dabei ist das Hervorbringen von allerlei kleinen Melodien nun einmal ein nicht zu vernachlässigender Anreiz. Unser MSX-Computer benutzt hierbei einen speziell dazu entwickelten 'Ton-Chip', d.h. unser Computer verfügt über einen besonderen Chip, mit dem Töne - und zwar in drei Stimmen (drei unabhängige Tonkanäle) - hervorgerufen werden können. Hierdurch haben wir beispielsweise die Möglichkeit, dreistimmige Akkorde zu spielen. Wir werden in der Folge die Anweisungen BEEP, PLAY und SOUND besprechen.

BEEP BEEP bedeutet soviel wie 'kurz piepen'. Wir gebrauchen BEEP nur als Warnungssignal, um beispielsweise dem Anwender deutlich zu machen, daß eine Berechnung vollzogen ist. Man stelle sich beispielsweise vor, daß der Computer eine lange Berechnung ausführen muß. Es ist langweilig, wenn Sie dann die ganze Zeit auf den Bildschirm starren müssen, um endlich zu sehen, wenn das Resultat erscheint. Es ist doch viel angenehmer für Sie, wenn Sie inzwischen beispielsweise irgendetwas nachlesen und erst nach der Erklingen des Piepsignals einen Blick auf Ihren Bildschirm werfen müssen.
Nachstehendes Programm illustriert dies:

```
10 PRINT "BEGINN"
20 FOR K=1 TO 1000
```

```
30 PRINT K
40 NEXT K
50 BEEP
60 PRINT "FERTIG"
```

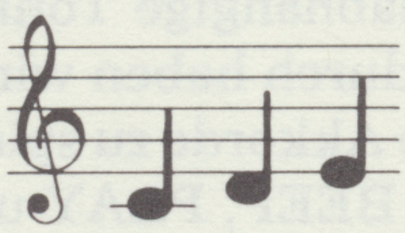
Wir sehen, daß die BEEP-Anweisung hier an das Ende des Programmes gesetzt ist. Man hört einen kurzen Piepton, sobald der Computer seine Rechenarbeit abgeschlossen hat.

PLAY Die PLAY-Anweisung ist ganz auf eine einfache Wiedergabe von Melodien ausgerichtet.
Es ist eine außerordentlich kraftvolle Anweisung, mit der wir u.a. folgendes angeben können:
- das Tempo, in dem gespielt werden soll
- die Oktave, auf die sich eine Note bezieht
- die Länge einer Note
- die Note (Ton) selbst
- Pausentakte
- das Volumen, in dem eine Note gespielt wird
- bestimmte Geräuscheffekte

PLAY kann als Anweisung wie auch als Befehl angewendet werden.
Geben Sie z.B. einmal ein:

```
PLAY "CDE"
```

Nach Betätigung von RETURN hören wir:



Kurzum, die Noten C, D und E, die zur Oktave mit dem G-Schlüssel gehören; wir werden diese Oktave im weiteren Verlauf als 'Oktave 4' bezeichnen.
Das folgende kleine Programm wendet die PLAY- Anweisung an, um aus unserem MSX-Computer eine einfache Kleinorgel zu machen.

```
10 A$=INKEY$
20 PLAY A$
30 GOTO 10
```

In Zeile 10 wird A$ die momentan eingedrückte Taste zugewiesen, die anschließend in der PLAY-Anweisung gebraucht wird usw.
Unsere Kleinorgel ist natürlich noch nicht perfekt, denn wir können u.a. das Tempo noch gar nicht regeln. Glücklicherweise gibt es verschiedene Regelmöglichkeiten.

Mehrere Tonkanäle

Wenn wir hinter PLAY die drei durch Kommas voneinander getrennten Strings eingeben, so gebraucht der Computer jede String für einen besonderen Tonkanal. Beispielsweise bei

```
PLAY "CDE"
```

wird immer von nur einem Kanal Gebrauch gemacht, bei

```
PLAY "CDE", "EFG"
```

werden zwei Kanäle gleichzeitig gebraucht und bei

```
PLAY "CDE", "EFG", "GAC"
```

werden die drei Kanäle gleichzeitig gebraucht. Dies bedeutet, daß die Noten C,E und G, anschließend D, F und A und schließlich E, G und C jeweils gleichzeitig zu hören sind.

Tempo

Das Tempo geben wir mit einem sogenannten T-Code an. Geben Sie einmal den folgenden Befehl:

```
PLAY "T200CDE"
```

Die Noten C, D und E werden nun in einem rascheren Tempo als zuvor gespielt. Dies ist der Fall, weil der Computer ohne genauere Mitteilung die 'Position' T120 hat, und T200 gibt damit ein rascheres Tempo an. Der T- Wert kann zwischen 32 und 256 liegen.

Lautstärke

Die Lautstärke (Volumen) geben wir mit einem V-Code an. Zur Illustration:

```
PLAY "T200V13CDE"
```

Wir hören nun die gleiche Melodie wie zuvor, nur ist die Lautstärke hörbar stärker eingestellt. Auch hier ist es so, daß der Computer ohne genauere Angaben in 'Position' V8 steht, und V13 entspricht damit einer größeren Lautstärke. Der V-Code kann zwischen dem Minimum 0 und dem Maximum 15 variieren.

Die Noten und die Oktave

Die Noten geben wir mit den Buchstaben C, D, E, F, G, A und B* an. Bei halben Tönen, beispielsweise Cis, gebrauchen wir das Zeichen # (oder +).
Beispiel:

```
PLAY "CC#" und ebenso PLAY "CC+"
```

so daß wir nacheinander C und Cis hören. Nun stellt sich noch

* Die in Deutschland als H bezeichnete Note wird international als B bezeichnet, während die Note B international als Bes bezeichnet wird. Bei B bekommen Sie also ein H zu hören und bei Bes ein B.

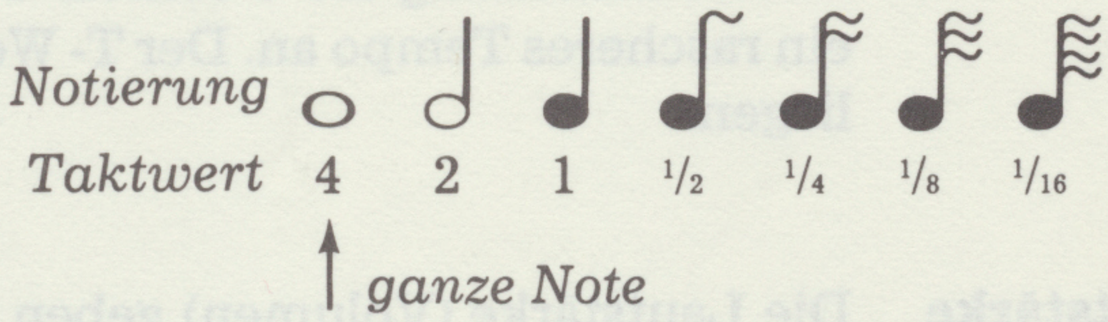
die Frage, wie wir das hohe C angeben. Ganz einfach, das hohe C gehört zur nächsten Oktave. Wir haben zuvor bereits erwähnt, daß der Computer ohne genauere Angabe von der Oktave 4 (O4) ausgeht. Man kann auch das Zeichen – gebrauchen, um einen halben Ton niedriger anzugeben.
Um das hohe C hervorzubringen, müssen wir also O5 angeben.
Beispiel:

```
PLAY "CDEO5CDE"
```

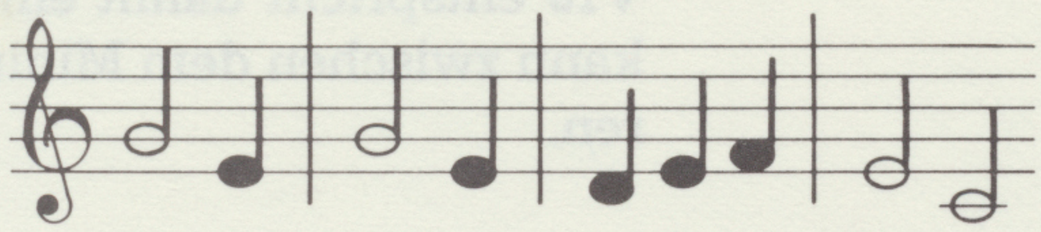
Hierbei hören wir zweimal die Reihe CDE, aber bei der letzten Reihe wird die Oktave höher gespielt.
Der Oktavenwert (O-Kode) kann minimal 1 und maximal 8 sein, und hieraus ist erkennbar, daß unser MSX-Computer Musik über acht Oktaven spielen kann.

Dauer der Noten

Bisher erklangen alle Noten in gleicher Länge. Wenn wir aber ein Notenbuch aufschlagen, so sehen wir, daß die Noten eine unterschiedliche Länge haben können.
In der Notenschrift geben wir dies an, indem wir das Notenköpfchen ausfüllen oder nicht und den Notenhals mit kleinen Wimpeln versehen können.



Den Notenwert geben wir mit dem L-Kode an. Ohne genauere Kennzeichnung geht der Computer von L4 aus, und dies entspricht einer ganzen Note.
Nachstehend illustrieren wir den L-Kode mit der holländischen Melodie 'boer, wat zeg je van m'n kippen'.



In einen PLAY-Befehl umgesetzt wird dies:

```
PLAY "L2GL4EL2GL4EDEFL2EL4C"
```

Dies können wir auch kürzer schreiben, indem wir sofort hinter einer Note den Wert angeben, also in unserem Beispiel:

```
PLAY "G2EG2EDEFE2C"
```

Pausenzeichen In der Notenschrift stoßen wir auch auf Pausenzeichen. Sie geben an, daß in einer bestimmten Zeit oder, besser gesagt, einer Anzahl von Takten kein Ton gespielt werden darf. Hierfür gebrauchen wir den R-Kode.
Nachstehende Übersicht zeigt uns die Zeichen, die in der Notenschrift hierfür gebraucht werden, zugleich auch den entsprechenden R-Kode.

Zeichen	—	—	}	˥	˥	˥	˥
Taktwert	4	2	1	1/2	1/4	1/8	1/16
R-Kode	1	2	4	8	16	32	64

Nachstehendes Beispiel illustriert dies:

```
10 PLAY "CDER4DECDECR2": GOTO 10
```

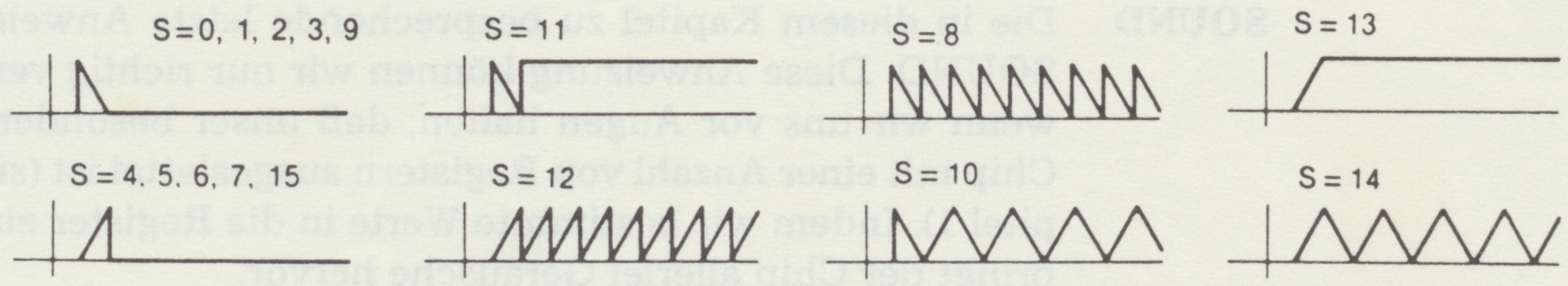
Nach dem ersten E ist eine Pause von einem Takt und nach dem letzten C eine Pause von 2 Takten. Wenn PLAY auf diese Weise in ein Programm aufgenommen wird, kann man den Takt der Musik noch besser studieren.
Wir unterbrechen die 'reine Weise' mit CTRL/STOP. In diesem Abschnitt über Pausezeichen wollen wir schließlich noch anmerken, daß wir hinter eine Note einen Punkt setzen dürfen: diese Note klingt dann anderthalb mal länger, z.B.:

```
PLAY "CD.E"
```

Auch hiermit können wir also den Rythmus der Musik bestimmen.

S- und M-Kode Die beiden letzten Kodes, die wir zur PLAY-Anweisung benutzen können, sind der S- und der M-Kode. Der S-Kode bezieht sich auf eine gewisse Klangfarbe, die wir damit geben können, und der M-Kode bestimmt, inwiefern das Muster der Klangfarbe wiederholt werden soll.
Der S-Kode gibt geradezu an, wie unser Ton nach einem bestimmten Muster verstärkt und abgeschwächt werden soll. Hierbei gebrauchen wir folgende Abbildungen:

5 = 0,1,2,3



Der M-Kode gibt dann die Länge des Zyklus wieder, in dem unsere mit S gewählte Form wiederholt werden soll. Ohne genauere Kennzeichnung geht der Computer bei S vom Wert 1 und bei M vom Wert 255 aus (M kann zwischen 1 und 65535 liegen). Beispiele:

```
PLAY "S10M510CDE"
```

und

```
PLAY "S8M6000CDE"
```

Beim ersten Beispiel hören wir bei jedem Ton einen auffällig stoßenden Klang; beim zweiten Beispiel Klingen die Töne, wie auf einem Klavier gespielt. Probieren Sie nun einmal selbst verschiedene Kombinationen, um die verschiedenen Möglichkeiten der S- und M-Kodes zu entdecken.

Ein ausgearbeites Beispiel

Im folgenden Beispiel spielt der Computer den Kanon 'Vater Jakob'. Man kann sehen, wie die Strings, die in der PLAY-Anweizung angewandt werden, zuvor mit String- Variablen aufgebaut werden.

```
 10 CLEAR 500
 20 A$="L4CDECR64CDEC"
 30 B$="EFG2EFG2"
 40 C$="L8GAGFL4ECL8GAGFL4EC"
 50 D$="C03G04C2C403G04C2"
 60 W$=A$+B$+C$+D$
 70 W1$=W
 80 W2$="R1R1"+W$
 90 W3$="R1R1"+W2$
100 PLAY W1$,W2$,W3$
```

(Anmerkung: In der String Zeile 50 werden die Großbuchstaben O verwendet.)

Die CLEAR-Anweisung wird angewandt, um ausreichend Platz zur String-Speicherung zu reservieren. In den Zeilen 20, 30, 40 und 50 wird die Melodie in kennzeichnenden Teilen angegeben. Zeile 60 gibt die Kombination von Strings an und damit die gesamte Melodie. Die Zeilen 70 und 80 bestimmen die zweite und dritte Stimme.

SOUND

Die in diesem Kapitel zu besprechende letzte Anweisung ist SOUND. Diese Anweisung können wir nur richtig verstehen, wenn wir uns vor Augen halten, daß unser besonderer Ton-Chip mit einer Anzahl von Registern ausgestattet ist (siehe Kapitel 1). Indem wir bestimmte Werte in die Register eingeben, bringt der Chip allerlei Geräusche hervor.

Unser Ton-Chip ist mit 13 Registern ausgestattet, die folgende Bedeutung haben:

Register-nummer	*wird gebraucht für*
0,1	Die Kombination des Inhalts von Register 0 und 1 bestimmt die Tonfrequenz von Kanal A.
2,3	Wie oben, jedoch für Kanal B.
4,5	Wie oben, jedoch für Kanal C.
6	Bestimmt die Klangfarbe eines rauschähnlich hervorzurufenden Klanges. Der Wert kann zwischen 0 bis 31 variieren.
7	Bestimmt, ob ein bestimmter Kanal ein Rauschen oder einen Ton hervorbringt.
8	Bestimmt die Lautstärke von Kanal A (Wert zwischen 0 und 15).
9	Wie 8, jedoch nun für Kanal B.
10	Wie 8, jedoch nun für Kanal C.
11,12	Bestimmt die Klangfarbe, und zwar die sogenannte Modulation eines Tones.
13	Bestimmt die Klangfarbe (Anschwellen und Ausklingen)

Zur Berechnung der Frequenz eines bestimmten Tones gebrauchen wir folgendes Hilfsprogramm:

```
10 INPUT "GEWUENSCHTE FREQUENZ"; F
20 A=1789772.5
30 B=INT(A/(16*F))
40 C=INT(B/256)
50 D=B-C
60 PRINT D,C
```

Die Werte D und C sind dann die Werte, die in das erste und zweite Register zu setzen sind, um den betreffenden Ton zu erzeugen.
Nachstehende Tabelle zeigt einige häufige Frequenzen.

Zahl	*Ton*	*Zahl*	*Ton*
110	A	440	a'
117	ais, bes (ais, b)	466	ais',bes' (ais', b')
123	B (H)	494	b' (h')
131	c	523	c''
139	cis, des	554	cis'', des''
147	d	587	d''
156	dis, es	622	dis'',es''
165	e	659	e''

Zahl	*Ton*	*Zahl*	*Ton*
175	f	698	f''
185	fis, ges	740	fis'',ges''
196	g	784	g''
208	gis, as	831	gis'', as''
220	a	880	a''
233	ais, bes (ais, b)	932	ais''',bes''' (ais''', b''')
247	b (h)	988	b'' (h'')
262	c'	1047	c'''
277	cis',des'	1109	cis''',des'''
294	d'	1175	d'''
311	dis',es'	1245	dis''',es'''
330	e'	1319	e'''
349	f'	1397	f'''
370	fis',ges'	1480	fis''',ges'''
392	g'	1568	g'''
415	gis',as'	1661	gis''',as'''
		1760	a'''

Beispiele Gesetzt den Fall, wir wollen A hervorrufen (Frequenz 440). Unser Programm produziert die Werte:

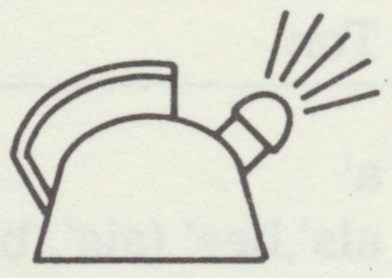
254 0

Dies führt dann zu folgenden SOUND-Anweisungen:

```
10 SOUND 0,254
20 SOUND 1,0
30 SOUND 8,11
```

Wir hören nun Ton A (auch versuchen mit PLAY'A'). Den Ton können wir unterbrechen mit CTRL/STOP.

Rauschen Wir werden die SOUND-Anweisung in der Praxis zum Hervorrufen allerlei auffallender Geräuscheffekte gebrauchen, z.B. Sirenengeheul und Knall-Geräusche. Folgende Beispiele geben einige Möglichkeiten:



```
10 REM PFEIFKESSEL
20 FOR K=255 TO 0 STEP-1
30 PRINT K
40 SOUND 0,K
50 SOUND 1,0
60 SOUND 8,10
70 NEXT K
```

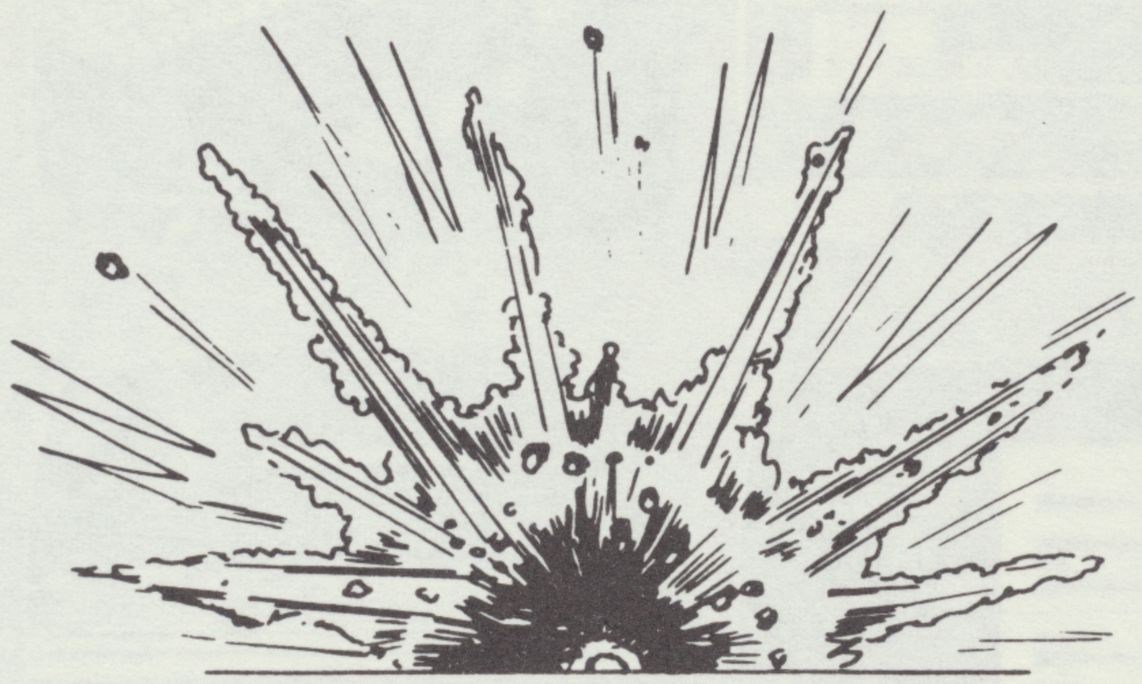
Wird Zeile 30 weggelassen, dann bekommt man eine beschleunigte Variante des Geräusches.
Folgendes Programm hat den Geräuscheffekt eines sterbendes Packmanns.

```
10 REM STERBENDER PACKMANN
20 FOR K=70 TO 100
30 SOUND 0,K
40 SOUND 1,0
50 SOUND 8,10
60 NEXT K
70 SOUND 8,0
```

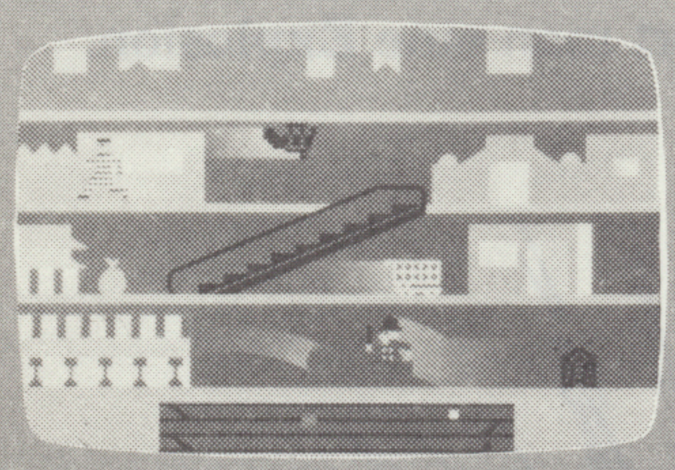
Mit dem folgenden Beispiel können wir noch verschiedene Effekte ausprobieren:

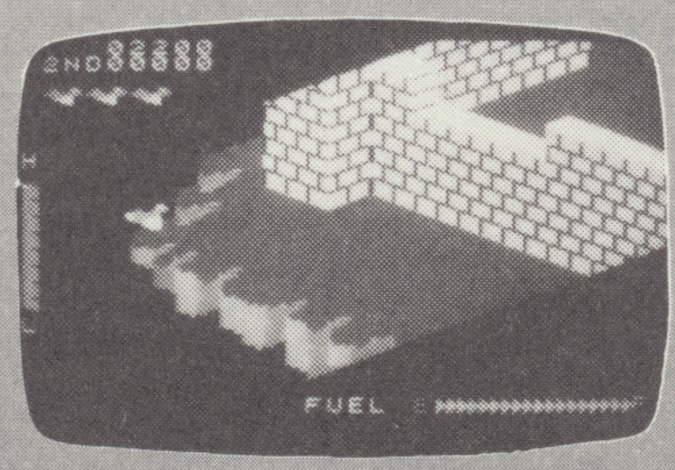
```
10 INPUT K,L,M
20 SOUND 0,250:SOUND 1,0
30 SOUND 6,K:SOUND 7,L:SOUND 13,M
40 FOR J=15 TO 0 STEP -0.05
50 SOUND 8,J
60 NEXT J
```

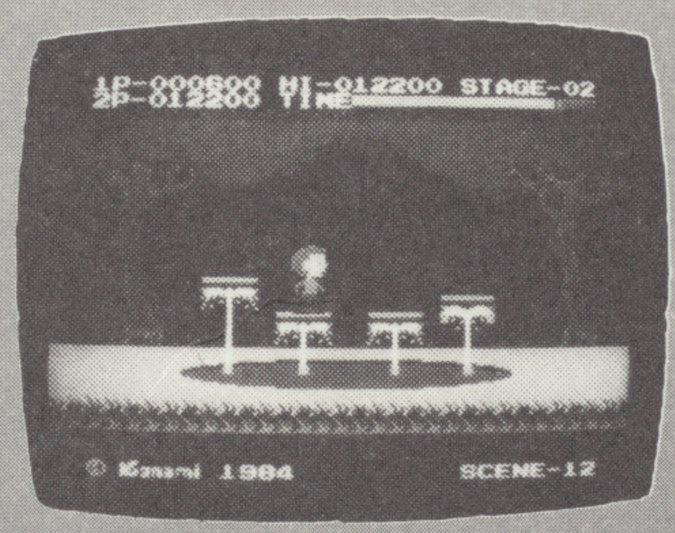
Bei K=10, L=10 und M=10 ergibt dies einen absterbenden Ton, aber bei K=20, L=20 und M=20 hört man eine Explosion.

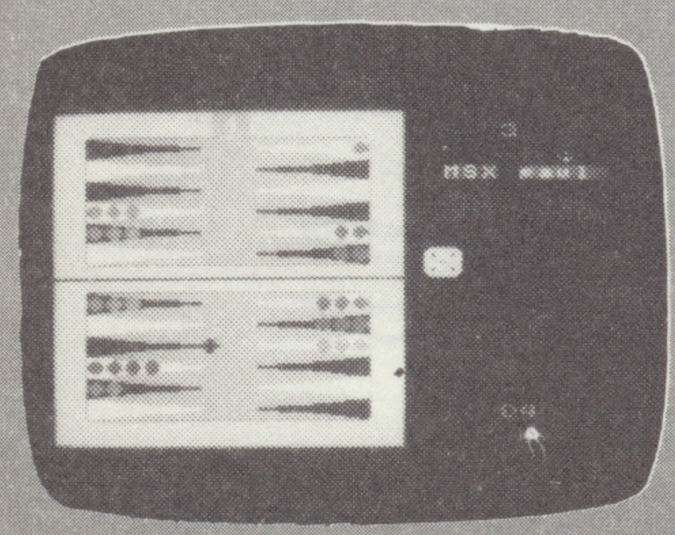


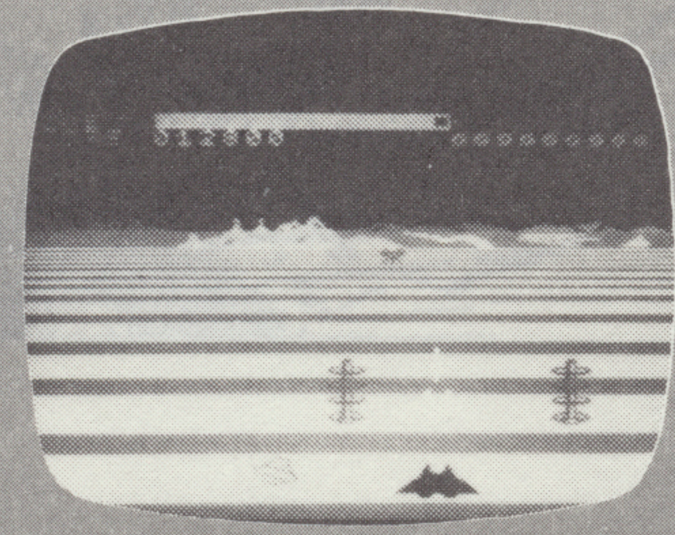
In diesem Kapitel besprechen wir die Möglichkeit des Arbeitens mit 'sprites'. Ein Sprite ist eine kleine Figur, die wir selbst definieren können. Dies geschieht mit der SPRITE-Anweisung.
Anschließend können wir mittels der PUT SPRITE-Anweisung diese Figur an einer beliebigen Stelle auf den Bildschirm bringen.
Wenn wir die Koordinaten eines Sprite auf bestimmte Weise verändern, wird dadurch bewirkt, daß sich die Figur über den Bildschirm bewegt. Mit einer ON SPRITE GOSUB-Anweisung können wir in eine beliebige Subroutine springen, wenn sich zwei Sprites überlappen (zusammenstoßen).

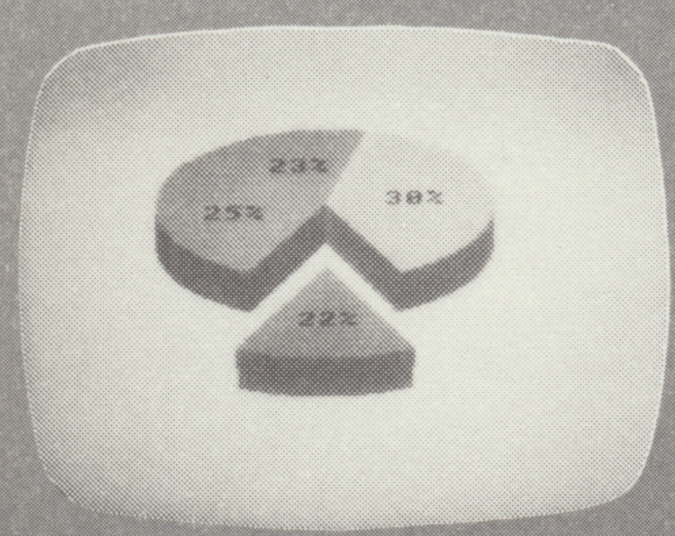


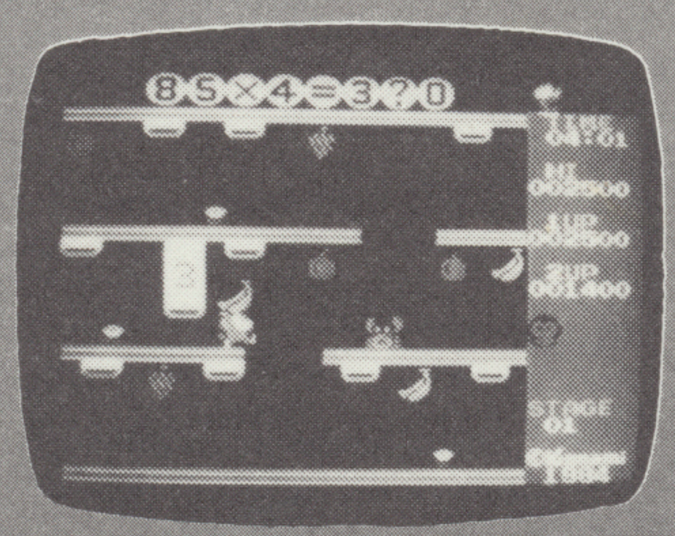


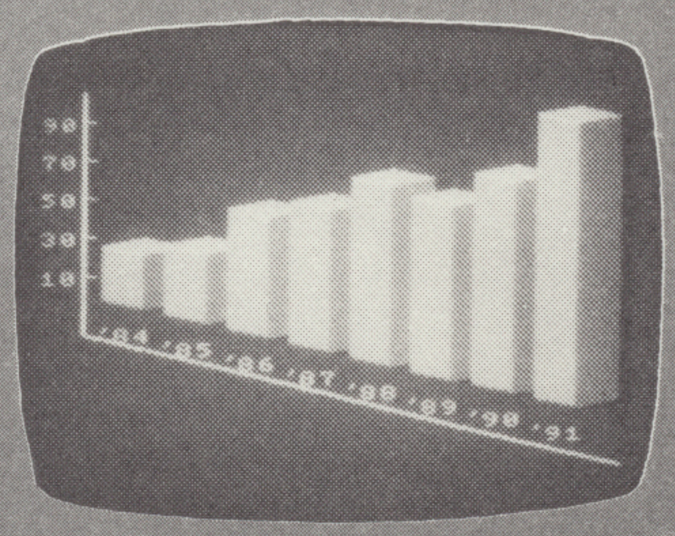


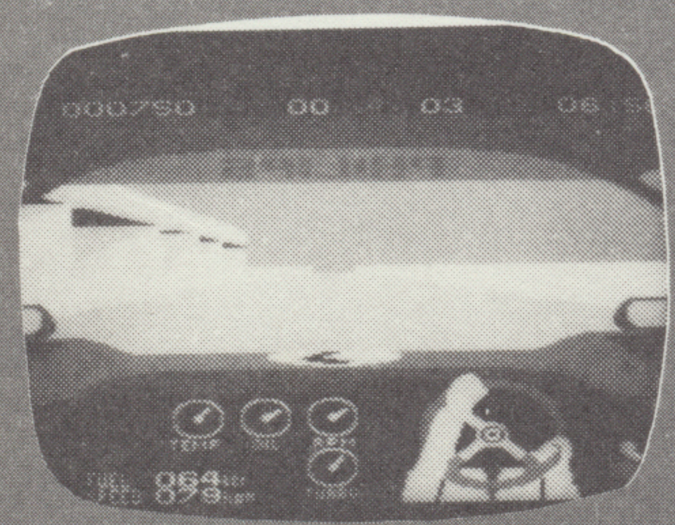


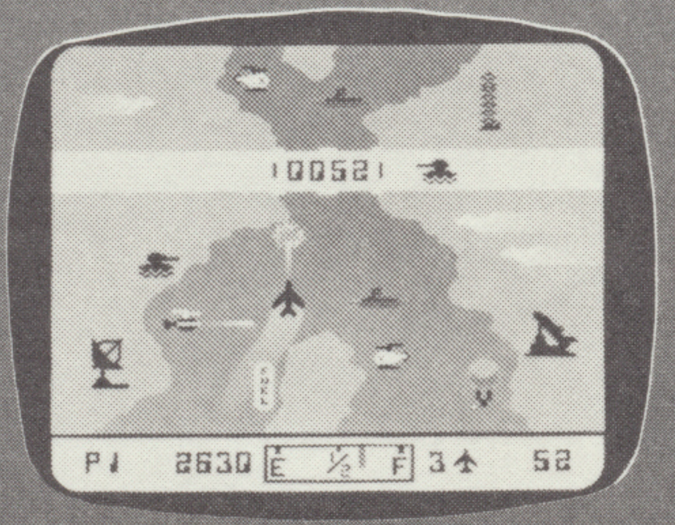


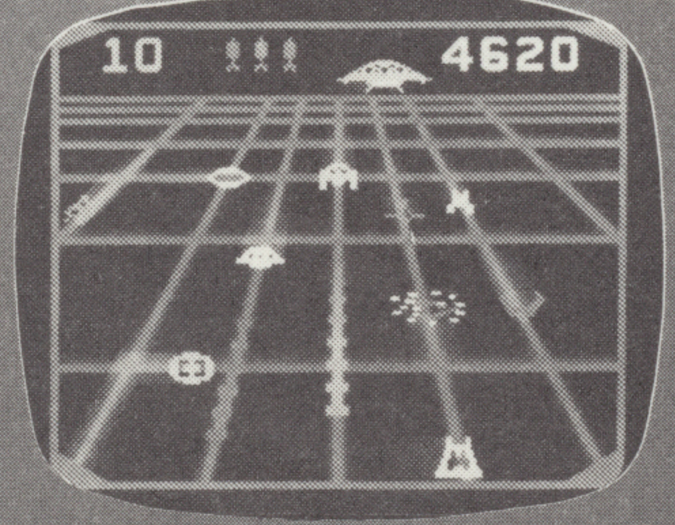












19. Sprites

Sprite

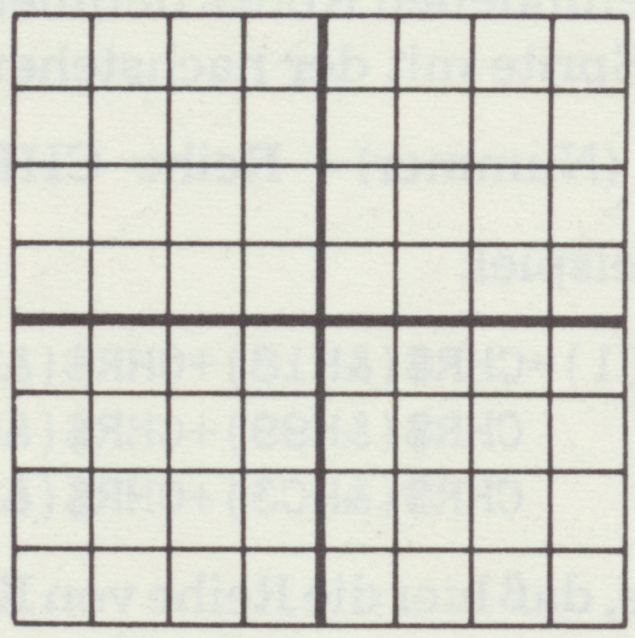
Als erstes stellt sich natürlich die Frage, was ein 'Sprite' eigentlich ist. Ein Sprite ist eine kleine Figur, die wir mit Hilfe eines Gittermusters festlegen können. Solch einer Figur geben wir anschließend eine Nummer. Danach können wir durch eine spezielle Anweisung mit solch einer Figur allerhand Handlungen vornehmen. Die wichtigste Manipulation ist es, daß wir solch eine Figur einfach über den Bildschirm bewegen lassen können.

Mit einer PRINT- und PSET-Anweisung können wir natürlich auch allerlei Figuren definieren. Zum Versetzen von mit PRINT- oder PSET-Anweisung aufgebauten Figuren sind jedoch relativ viele Anweisungen nötig, so daß ein schnelles Versetzen solcher Figuren eigentlich nicht möglich ist.

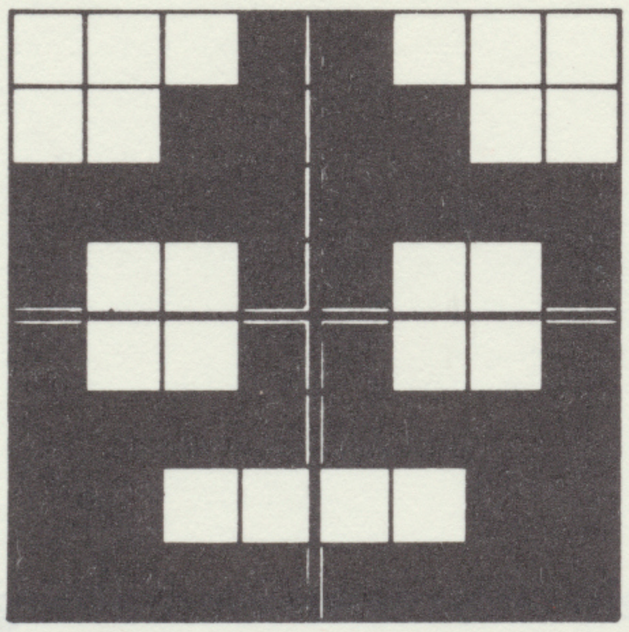
Mit Sprites haben wir also sehr viel mehr Möglichkeiten. So haben wir beispielsweise damit ein praktisches Gerät, um Packmann-artige Wesen über den Bildschirm irren zu lassen.

Festlegung eines Sprite

In der Folge werden wir aufzeigen, wie wir solch eine Figur, also ein Sprite, definieren können. Hierbei gehen wir von einer 8x8-Matrix aus.



Anschließend färben wir dann bestimmte Fächer schwarz, um die von uns gewünschte Figur zu bekommen.
Um beispielsweise einen Geist zu bekommen, können wir die Fächer folgendermaßen einfärben:



0001 : 1000	18
0011 : 1100	3C
1111 : 1111	FF
1001 : 1001	99
1111 : 1111	FF
1100 : 0011	C3
1111 : 1111	FF

Links sehen wir den Geist, und in der linken Spalte sieht man, wie wir mit Hilfe von Nullen und Einsen (Weiß und Schwarz) dieselbe Figur zeichnen können.
Die punktierte Spalte ist eine 'Hilfslinie'; sie teilt jede Zeile auf in zwei Reihen von vier Einsen und/oder Nullen. Die oberste Zeile besteht beispielsweise aus den Reihen 0001 und 1000.
In der rechten Spalte ist eine Schreibweise mit anderen Kodes zu sehen. Diese Kodes kann man direkt aus der folgenden Tabelle ableiten. (Anmerkung: es handelt sich hier um die sogenannte Hexadezimal-Kodierung, die im Kapitel 8 besprochen wurde.)

Reihe	*Kode*	*Reihe*	*Kode*	*Reihe*	*Kode*	*Reihe*	*Kode*
0000	0	0100	4	1000	8	1100	C
0001	1	0101	5	1001	9	1101	D
0010	2	0110	6	1010	A	1110	E
0011	3	0111	7	1011	B	1111	F

Mit den so gefundenen Kodes definieren wir nun in einem Programm das Sprite mit der nachstehenden Anweisung:

```
SPRITE$ (Nummer)= Reihe CHR$-Anweisungen
```

Für unser Beispiel:

```
SPRITE$(1)=CHR$(&H18)+CHR$(&H3C)+CHR$(&HFF)+
           CHR$(&H99)+CHR$(&H99)+CHR$(&HFF)+
           CHR$(&HC3)+CHR$(&HFF)
```

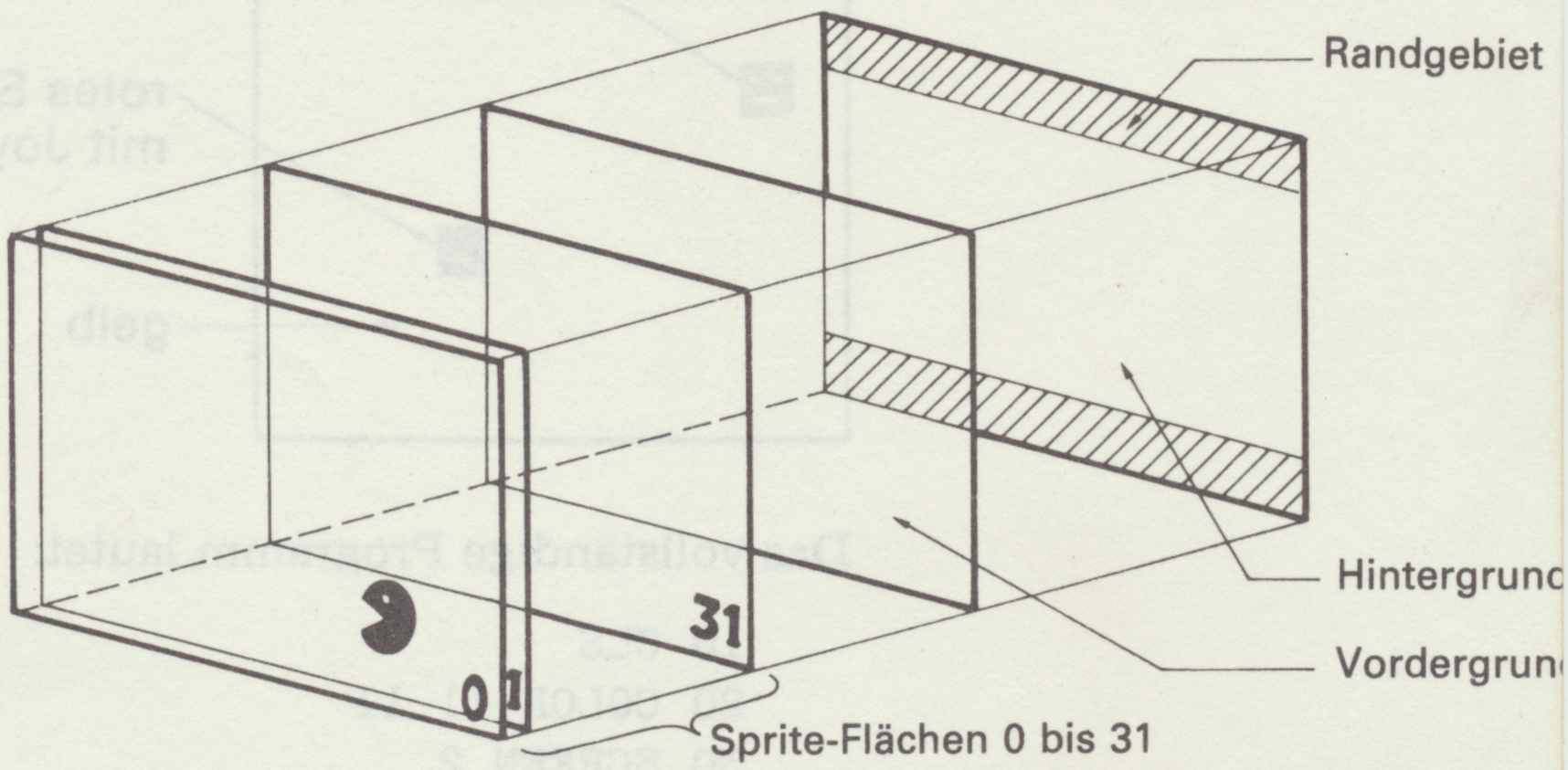
Beachten Sie, daß hier die Reihe von Kodes 18,3C,FF usw. in die CHR$-Anweisungen aufgenommen worden ist, wobei an jedem Kode das Zeichen &H vorausgeht.

PUT SPRITE Auf die vorstehende Weise ist unser Sprite (Nummer 1) völlig festgelegt, und wir können anschließend die Figur auf einen beliebigen Platz des Bildschirms setzen. Dazu verwenden wir die PUT SPRITE-Anweisung in der folgenden Form:

`PUT SPRITE` Fläche, (x,y), Farbe, Sprite-Nummer

und hierin ist:

Fläche eine Nummer zwischen 0 und 31. Man denke hierbei an ein folgendes Schema:



Die Sprites bewegen sich gewissermaßen über durchsichtige Glasschirme, die auf oben dargestellte Weise hintereinander aufgestellt sind. Die Lage einer Fläche bestimmt, welches Sprite abgedeckt wird, wenn sich zwei Sprites überschneiden.

(x,y) Die Koordinaten der Stelle, an der ein Sprite abgebildet werden soll.

Farbe Der Farbkode des Sprite

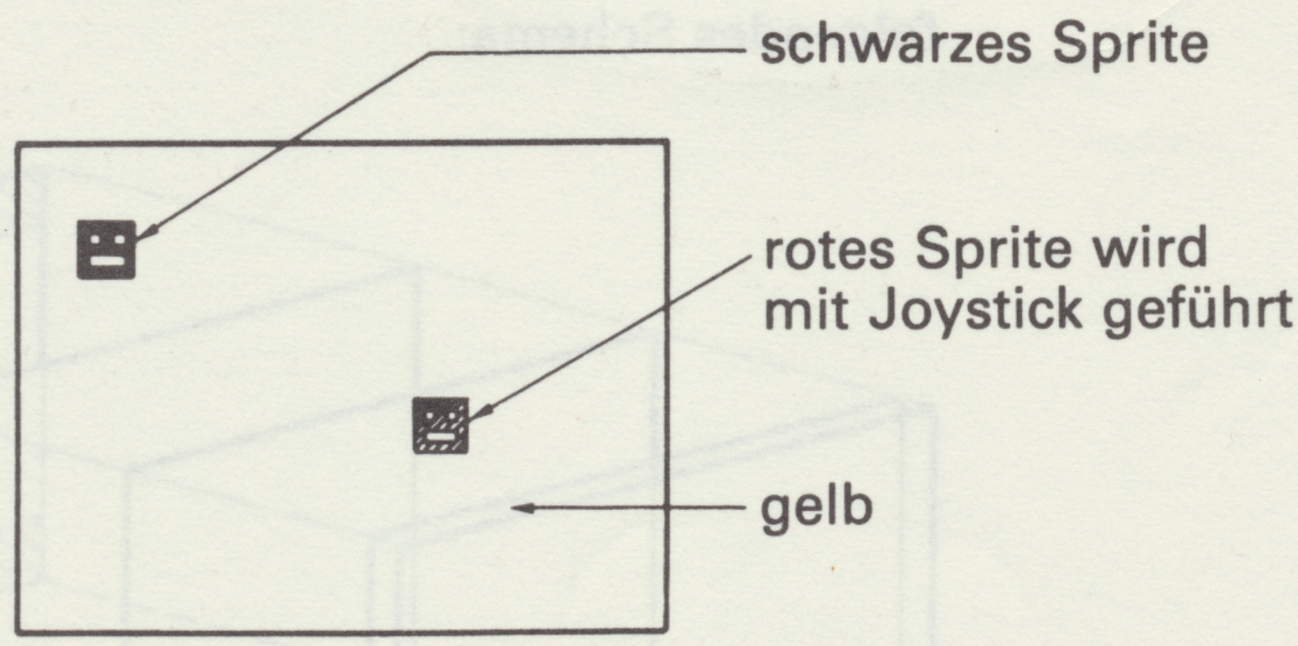
Nummer Die Nummer, die ein Sprite auf Grund der SPRITE$(Nummer)-Anweisung bekommen hat.

ON SPRITE GOSUB und SPRITE ON Bei vielen Spielen ist es wichtig zu wissen, ob sich zwei Sprites in einem gegebenen Augenblick überdecken oder, volkstümlicher gesagt, ob sie zusammenstoßen.
Dies können wir bei MSX-BASIC mit der ON SPRITE GOSUB-Anweisung feststellen. Sie hat immer die folgende Form:

`ON SPRITE GOSUB` Zeilennummer

Wenn der Computer nun feststellt, daß sich zwei Sprites überschneiden, dann wird über die angegebene Zeilennummer zu einer Subroutine gesprungen.
Dazu müssen wir allerdings erst die Aufmerksamkeit des Computers erregen. Dies geschieht mit der SPRITE ON-Anweisung.

Ein Beispiel Das folgende Programm zeigt uns ein Beispiel. Auf den Bildschirm kommen zwei Geister. Der eine (schwarze) Geist steht still, während der andere (rote) Geist mit dem Joystick gesteuert wird. Wir haben den Steuermechanismus bereits im Kapitel 15 besprochen.
Das Bild:



Das vollständige Programm lautet:

```
 10 CLS
 20 COLOR,11,11
 30 SCREEN 2
 40 X=100:Y=100
 50 FOR K=1 TO 2
 60 SPRITE$(K)=CHR$(&H18)+CHR$(&H3C)+CHR$(&HFF)+
                CHR$(&H99)+CHR$(&H99)+CHR$(&HFF)+
                CHR$(&HC3)+CHR$(&HFF)
 70 NEXT K
 80 SPRITE ON
 90 PUT SPRITE 0,(40,40),1,1
100 D=STICK(1)
110 IF D=1 THEN Y=Y-1
120 IF D=2 THEN X=X+1:Y=Y-1
130 IF D=3 THEN X=X+1
140 IF D=4 THEN X=X+1:Y=Y+1
150 IF D=5 THEN Y=Y+1
160 IF D=6 THEN X=X-1:Y=Y+1
170 IF D=7 THEN X=X-1
180 IF D=8 THEN X=X-1:Y=Y-1
190 PUT SPRITE 1,(X,Y),6,2
200 ON SPRITE GOSUB 220
210 GOTO 100
220 PLAY "CCC"
230 GOTO 230
240 RETURN
```

Die Zeilen 10 bis 30 sprechen für sich. Zeile 40 bestimmt die Ausgangsposition des roten Sprite. Hiernach werden in den Zeilen 50 bis 70 zwei Sprites definiert, die übrigens genau die gleiche Form haben (die ja bereits besprochen wurde).
Anschließend wird der Computer mit der SPRITE ON- Anweisung (Zeile 80) in den Zustand 'Zusammenstöße beachten' gebracht. Zeile 90 setzt das schwarze Sprite auf den Bildschirm, und zwar auf die entsprechende Position (40,40). Anschließend sehen wir die bekannte Konstruktion zur Übermittlung der Joystick-Position. Diese Stellung wird dann der PUT SPRITE-Anweisung von Zeile 190 mitgegeben. Hiermit wird das rote Sprite versetzt.
Zeile 200 zeigt die 'Kollisionsprobe': wenn ein Zusammenstoß stattfindet, springt der Computer zur Subroutine von Zeile 220.

Anmerkungen

Zum Arbeiten mit Sprites ist noch folgendes anzumerken:

- Wir können das Sprite um den Faktor 2 vergrößern, indem wir Zeile 30 durch SCREEN 2,1 ersetzen
- Wir können Sprites auch in einer Matrix 16x16 definieren. Ein derartiges Muster besteht daher aus vier Blöcken mit einer 8x8-Matrix. Diese Blöcke werden folgendermaßen numeriert:

1	3
2	4

Die einfachste Weise ist dann die Festlegung dieses Musters in vier Strings:

```
A$=CHR$( )+ usw.        (Definition Block 1)
B$=CHR$( )+ usw.        (Definition Block 2)
```

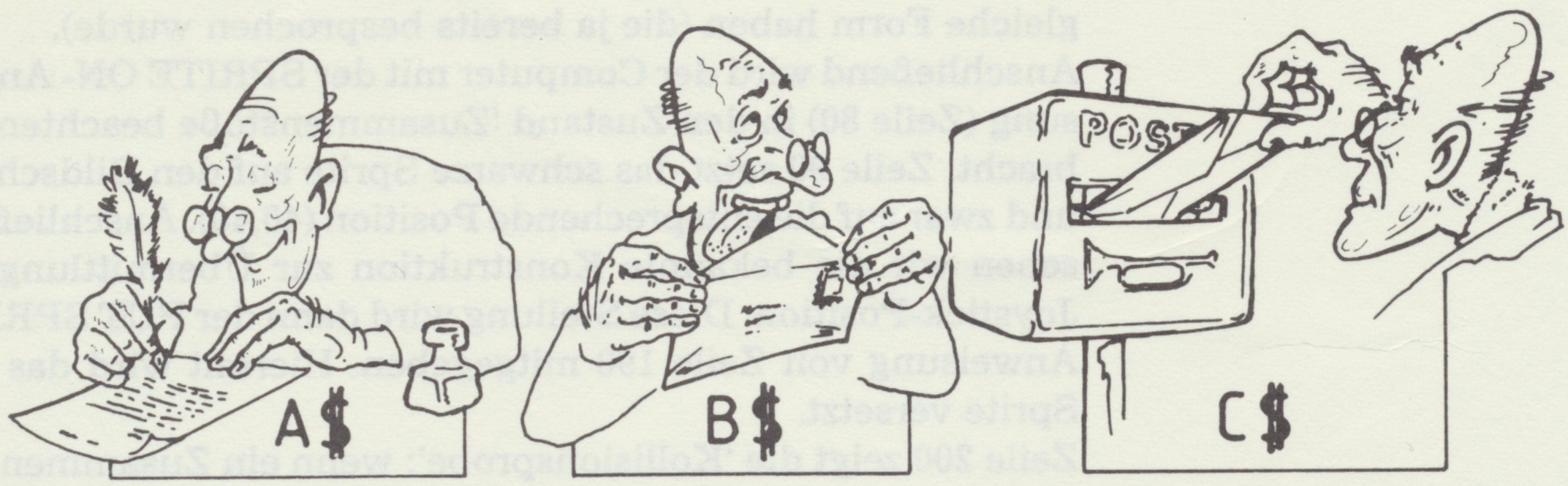
und ebenso C$ und D$, sowie anschließend

```
SPRITE$(1)=A$+B$+C$+D$
```

Falls man mit Sprites 16x16 arbeitet, wendet man die Anweisung SCREEN 2,2 oder SCREEN 2,3 an, sofern man eine Vergrößerung um den Faktor 2 wünscht.

- Maximal dürfen wir 256 Sprite-Muster mit einer 8x8 Matrix definieren und 64 Muster mit einer Matrix 16x16. Dabei ist jedoch zu berücksichtigen, daß auf jeder Sprite-Fläche nur ein Sprite wiedergegeben werden darf.
- Man darf höchstens 4 Sprite-Muster abbilden, die horizontal in einer Reihe stehen.

Anhang A Dateien



Einlesen Eine Datei (engl. file) ist eine gespeicherte Sammlung von Daten unter einem bestimmten Namen. Die Daten werden immer in einer bestimmten Reihenfolge gespeichert, und deshalb kann man die Daten auch wiederfinden. Die Reihenfolge ist eigentlich recht einfach: die Daten werden nacheinander (sequenziell) gespeichert.
Dies verdeutlicht ein Beispiel:

```
10 REM DEMO EINLESEN DATEI
20 A$="ADAM"
30 B$="BERTA"
40 C$="CAESAR"
50 OPEN "CAS:DATA" FOR OUTPUT AS#1
60 PRINT#1,A$
70 PRINT#1,B$
80 PRINT#1,C$
90 CLOSE#1
```

In diesem Beispiel werden die Strings A$,B$ und C$ gespeichert.
Dazu wird zuerst eine Datei geöffnet, d.h. wir definieren hier eine Datei durch Angabe eines Namens (DATA), geben dieser Datei ein Etikett(#) und kennzeichnen ferner, daß hier die Kassette (CAS) zur Speicherung benutzt wird.
Die übliche Form der OPEN-Anweisung ist:

OPEN "Name Speichermedium:[Name Datei]" FOR OUTPUT AS[Nummer der Datei].

Als 'Name Speichermedium' kann man verwenden:

CAS	Kassettenrecorder
CRT	Text-Bildschirm
GRP	Grafischer Bildschirm
LPT	Drucker
COM(n)	RS 232 C (siehe Anhang N)

'Name Datei' darf aus höchstens 6 Zeichen bestehen. Wenn die Datei erst einmal auf diese Weise gekennzeichnet ist, können wir darin Daten speichern. Dies geschieht mit der PRINT-Anweisung, hinter die nun die Zeichen #, gesetzt sind (Zeile 60).
Nach Beendigung muß, wie wir dies ausdrücken, 'die Datei geschlossen werden'. Dies geschieht mit der CLOSE-Anweisung in der folgenden Form:

CLOSE [#Nummer der Datei]
Wenn wir [#Nummer der Datei] weglassen, dann werden alle noch geöffneten Dateien geschlossen.

Einsetzen von Text auf dem grafischen Schirm

Auf diese Weise kann man auch Schriftzeichen auf den grafischen Bildschirm (GRP) setzen.
Die Anweisung ist dann beispielsweise:

```
OPEN "GRP:BILD" FOR OUTPUT AS#1
PRESET (50,50)
PRINT #1, "RESULTAT"
```

Mit der PRESET-Anweisung wird hier bestimmt, an welcher Stelle der Text erscheinen soll.

Abrufen

Beim Abruf einer Datei findet man fast die gleiche Konstruktion wie beim Speichern. Wiederum wird die Datei geöffnet, jedoch jetzt mit der Anweisung FOR INPUT, und statt der PRINT#,-Anweisung verwenden wir jetzt die INPUT#,-Anweisung. Folgendes Beispiel bezieht sich auf unser erstes Beispiel:

```
10 REM DEMO LESEN DATEI
20 OPEN "CAS:DATA" FOR INPUT AS#1
30 INPUT#1,A$,B$,C$
40 PRINT A$,B$,C$
50 CLOSE#1
```

Die PRINT-Anweisung von Zeile 40 wurde nur zur Kontrolle des Resultats aufgenommen. Wenn man nicht im voraus weiß, wieviel Daten eine Datei enthält, kann man folgende Konstruktion anwenden:

```
10 OPEN "CAS:DATA" FOR INPUT AS#1
20 IF EOF(1)=-1 THEN GOTO 60
30 INPUT#1, X$
40 PRINT X$
50 GOTO 20
60 CLOSE#1
```

Nun wird jeder Wert der Datei abgedruckt, und am Ende der Datei erfolgt ein Sprung zur Zeile 60.

Über das Komma (CR und LF)

Wenn wir eine PRINT-Anweisung ausgeführt haben, versetzt der Computer den Cursor an den Anfang einer neuen Zeile. Der Beginn einer Zeile wird mit der Abkürzung CR angegeben, und LF bezieht sich auf das Versetzen in eine neue Zeile. CR und LF haben jeweils einen spezifischen ASCII-Kode. Diese Zeichen werden zugleich auch als Trennungszeichen verwendet, wenn Daten mit PRINT# auf Kassette gespeichert werden.
Beispiel 1:

```
10 MAXFILES=1
20 OPEN "CAS:ADDR" FOR OUTPUT AS #1
30 A=15:B=23.5:C=10:D=25.2:E=13
40 PRINT #1, A; B; C
50 PRINT #1, D; E
60 CLOSE #1
```

In diesem Fall können die Daten auf der Kassette folgendermaßen ins Bild gebracht werden:

15	,	23.5	,	10	CR	LF	25.2	,	13	CR	LF

Vorsicht beim Einsetzen gesonderter Strings durch eine PRINT#-Anweisung. Hierbei müssen immer Kommas als Trennungszeichen eingesetzt werden.
Beispiel 2:

```
10 MAXFILES=1
20 OPEN "CAS:NAME" FOR OUTPUT AS #1
30 A$="JOHN":B$="ELLY"
40 PRINT #1, A$,",";B$
50 CLOSE #1
```

Im Bild:

JOHN	,	ELLY	CR	LF

Zum Abschluß noch folgende Hinweise:

- Es darf höchstens eine Datei zugleich geöffnet werden, sofern nicht mit einem MAXFILES-Befehl eine größere Anzahl angegeben wurde (z.B. MAXFILES=3).
- Statt PRINT, # kann auch die Form PRINT USING #, gebraucht werden.
- Statt INPUT #, darf auch die Form LINE INPUT #, gebraucht werden.

Anhang B Zeichen und ihre Kodes

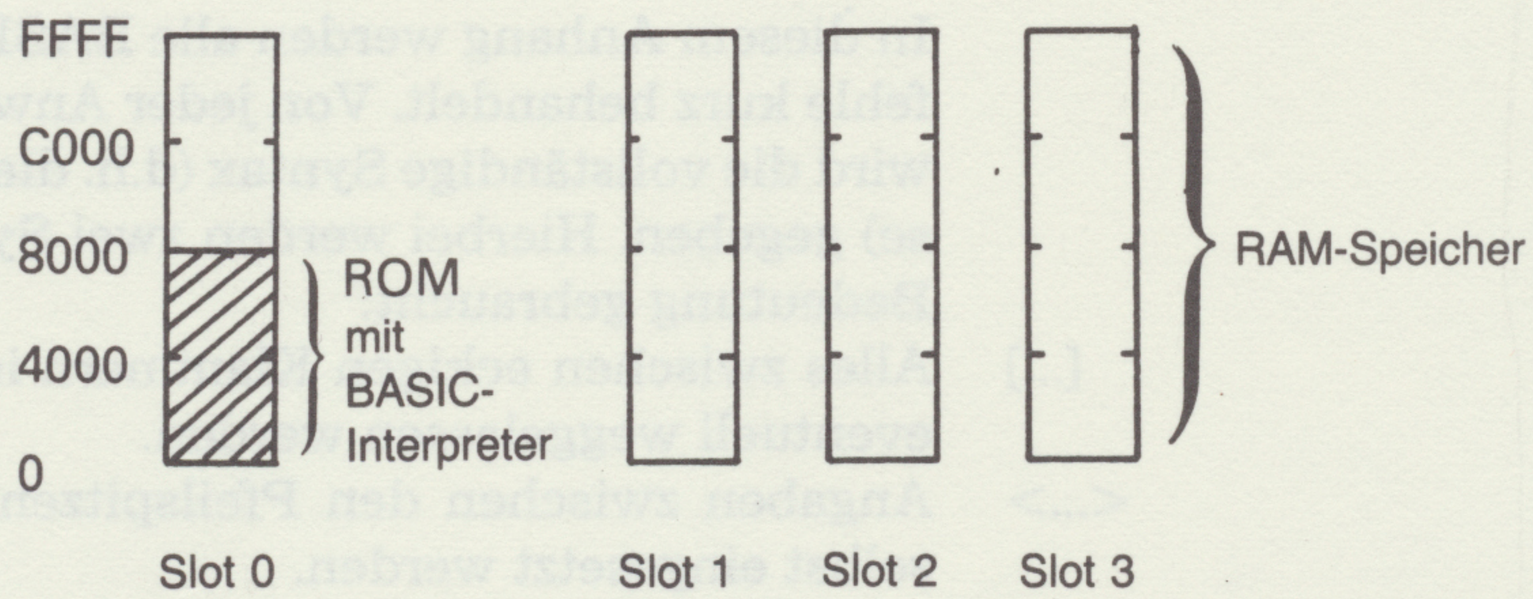
Die folgende Tabelle gibt eine Übersicht aller Zeichen und der zugehörigen Kodes.

Kode	Zeichen	Kode	Zeichen	Kode	Zeichen	Kode	Zeichen
0	Null	32	Abstand	64	@	96	`
1	☺	33	!	65	A	97	a
2	☻	34	"	66	B	98	b
3	♥	35	#	67	C	99	c
4	♦	36	$	68	D	100	d
5	♣	37	%	69	E	101	e
6	♠	38	&	70	F	102	f
7	•	39	'	71	G	103	g
8	◘	40	(	72	H	104	h
9	○	41	)	73	I	105	i
10	◙	42	*	74	J	106	j
11	♂	43	+	75	K	107	k
12	♀	44	,	76	L	108	l
13	♪	45	—	77	M	109	m
14	♫	46	.	78	N	110	n
15	☼	47	/	79	O	111	o
16	┼	48	Ø	80	P	112	p
17	┴	49	1	81	Q	113	q
18	┬	50	2	82	R	114	r
19	┤	51	3	83	S	115	s
20	├	52	4	84	T	116	t
21	┼	53	5	85	U	117	u
22	─	54	6	86	V	118	v
23	─	55	7	87	W	119	w
24	┌	56	8	88	X	120	x
25	┐	57	9	89	Y	121	y
26	└	58	:	90	Z	122	z
27	┘	59	;	91	[	123	{
28	╳	60	<	92	\	124	¦
29	╱	61	=	93	]	125	}
30	╲	62	>	94	^	126	~
31	┼	63	?	95	_	127	

Kode	Zeichen	Kode	Zeichen	Kode	Zeichen	Kode	Zeichen
128	Ç	160	á	192	▬	224	α
129	ü	161	í	193	▚	225	β
130	é	162	ó	194	▬	226	Γ
131	â	163	ú	195	▬	227	Π
132	ä	164	ñ	196	▪	228	Σ
133	à	165	Ñ	197	▬	229	σ
134	å	166	ª	198	▌	230	µ
135	ç	167	º	199	▞	231	γ
136	ê	168	¿	200	▬	232	Φ
137	ë	169	⌐	201	▌	233	Θ
138	è	170	¬	202	▬	234	Ω
139	ï	171	½	203	//	235	δ
140	î	172	¼	204	\\	236	∞
141	ì	173	¡	205	▼	237	Ø
142	Ä	174	«	206	▲	238	∈
143	Å	175	»	207	▶	239	∩
144	É	176	Ã	208	◀	240	≡
145	æ	177	ã	209	⧗	241	±
146	Æ	178	Ĩ	210	⧓	242	≥
147	ô	179	ĩ	211	▮	243	≤
148	ö	180	Õ	212	▪	244	⌠
149	ò	181	õ	213	▪	245	⌡
150	û	182	Ũ	214	▪	246	÷
151	ù	183	ũ	215	✠	247	≈
152	ÿ	184	Ĳ	216	△	248	°
153	Ö	185	ĳ	217	ǂ	249	•
154	Ü	186	¾	218	ω	250	⁻
155	¢	187	⌣	219	▬	251	√
156	£	188	◇	220	▬	252	η
157	¥	189	‰	221	▌	253	²
158	₧	190	¶	222	▌.	254	■
159	ƒ	191	§	223	▬	255	

Anhang C Speichereinteilung

Speichereinteilung



Oben sind vier Blöcke abgebildet. Jeder Block stellt ein sogenanntes 'Slot' dar. Solch ein Slot kennzeichnet eigentlich eine Erweiterungsmöglichkeit. So sind Slot 1 und 2 die Erweiterungsmöglichkeit, die wir an der Oberseite des Computers sehen. Slot 0 und 3 befinden sich im Computer. Slot 0 enthält den ROM-Speicher, in dem sich unser BASIC-Interpreter (siehe Kapitel 1) befindet. Slot 3 enthält den RAM-Speicher. Die waagerechte Unterteilung gibt den Adreßraum an. In diesem Fall werden die Adressen 0000-8000 vom BASIC- Interpreter eingenommen, und der übrige Teil wird vom RAM-Teil eingenommen. Wir sprechen in dieser Situation von Austausch der Speichergebiete oder, wie Computer-Fachleute es ausdrücken, von 'bank switching'. Wenn wir in Slot 1 oder 2 eine Cartridge stecken, wird beispielsweise die Abteilung des BASIC-Interpreters ausgeschaltet, und an deren Stelle kommt der ROM-Speicher der Cartridge.

Die vorerwähnten Slots werden als Primärslots bezeichnet. Jedes Primärslot kann wiederum durch 4 Sekundärslots erweitert werden. Der RAM-Speicher braucht nicht unbedingt im Primärslot 3 zu stecken. Je nach Ausführung Ihres MSX-Computers kann der RAM-Speicher sich auch in einem anderen Primärslot oder gar in einem Sekundärslot befinden.

Anhang D Übersicht der BASIC-Anweisungen und -Befehle

In diesem Anhang werden alle BASIC-Anweisungen und -Befehle kurz behandelt. Von jeder Anweisung und jedem Befehl wird die vollständige Syntax (d.h. die vollständige Schreibweise) gegeben. Hierbei werden zwei Symbole mit der folgenden Bedeutung gebraucht:

[...] Alles zwischen eckigen Klammern ist fakultativ, d.h. es kann eventuell weggelassen werden.

<...> Angaben zwischen den Pfeilspitzen müssen vom Anwender selbst eingesetzt werden.

Zur Verdeutlichung: in dieser Übersicht wird der Ausdruck 'Datei' wie 'File' gebraucht, um einen Datenbestand zu bezeichnen.

Bei einer Anzahl von Ausdrücken findet man drei Punkte, beispielsweise bei:

CALL <Name> [<String>,<String>...]

Die drei Punkte am Ende dieses Ausdruckes bedeuten:'Dieser Ausdruck kann auf gezeigte Weise fortgesetzt werden'. In diesem Fall können also weitere Strings, getrennt durch ein Komma, aufgenommen werden.

AUTO[<Zeilennummern>][,[<Schrittweite>]]

Generiert automatisch Zeilennummern während der Eingabe eines Programmes. Wenn keine <Zeilennummer> angegeben wird, wird mit Zeilennummer 10 begonnen, sonst mit der genannten <Zeilennummer>. Ist keine <Schrittweite> angegeben, wird die Zeilennummer jeweils um 10 erhöht, anderenfalls in der genannten <Schrittweite>.

Wird automatisch eine Zeilennummer generiert, die bereits gebraucht wurde, erscheint zur Warnung ein Sternchen *.

Der Befehl wird mit CRTL/C beendet, d.h. bei betätigter CTRL-Taste wird auch C gedrückt.

BEEP

Generiert einen kurzen Piepton.

BLOAD"<dev>:[<Dateiname>]"[,R][,<Verschiebung>]
Hiermit wird ein in Maschinenkode geschriebenes Programm geladen, das mit einem BSAVE-Befehl gespeichert ist.
Wird für den Ausdruck CAS eingesetzt, so wird damit der Kassettenrecorder angegeben. Der Ausdruck Dateiname bezieht sich auf den erteilten Namen (höchstens 6 Zeichen, beispielsweise PROG1). Wird ein R eingegeben, so wird das Programm nach dem Laden sofort gestartet.
Wird eine Verschiebung (ganze Zahl) angegeben, so wird die neue Startadresse gleich der alten + der Verschiebung.

BLOAD "<dev>:[<Dateiname>]",S
Wie der vorige Befehl, aber in diesem Fall weist S auf den Video-RAM-Speicher; beachten, daß <dev> hier nur eine Diskette sein kann.

BSAVE"<dev>:[<Dateiname>]",<Beginnadresse, Endadresse>[,<Beginnadresse für Start des Programmes>]
Hiermit wird ein im Maschinenkode geschriebenes Programm in einem externen Medium gespeichert. Der Ausdruck 'dev' und 'Dateiname' wurden unter BLOAD erklärt. Beginn- und Endadresse kennzeichnen das Speichergebiet, das in das externe Medium kopiert wird.

BSAVE"<dev>:[<Dateiname>]",<Beginnadresse>, Endadresse>,S
Wie oben, aber jetzt handelt es sich um den Video-RAM- Speicher. <dev> kann hier nur eine Diskette sein.

CALL <Name>[(<String>,<String>...)]
Allgemeiner Befehl zur Durchführung bestimmter Subroutinen, die im ROM-Cartridge gespeichert sind.

CIRCLE[STEP](<x,y>),<r>[,[<Farbe>][,[<Anfangswinkel>] [,[<Endwinkel>][,<Abplattung>]]]]
Generiert eine Ellipse oder (mit Abplattung 1.4) einen Kreis. Mit (x,y) wird der Mittelpunkt angegeben, r kennzeichnet den Radius. Man kann eventuell auch nur einen Teil zeichnen, indem man einen Beginn- und einen Endwinkel (im Bogenmaß angeben) nennt. Wird STEP weggelassen, so wird das übliche Koordinatensystem benutzt. Mit STEP verschiebt sich das Koordinatensystem auf den unter STEP genannten Punkt.

CLEAR [<Speicherraum für Strings>[,<höchste Adresse>]]
Hiermit werden alle Variablen gleich 0 gesetzt, und außerdem wird ein Speicherbereich für die Eingabe von Zeichen (Strings, Buchstaben) reserviert.
Die 'höchste Adresse' bestimmte die höchste Adresse zum Ge-

brauch in BASIC. Schließlich hat CLEAR zur Folge, daß alle eventuell noch offenen Dateien geschlossen werden. Es ist zu berücksichtigen, daß der Computer ohne CLEAR nur 200 Wörter zur Speicherung von Buchstaben und dergleichen reserviert.

CLOAD["<Dateiname>"]
Befehl zum Einlesen eines Programmes, eventuell unter einem Namen (max. 6 Buchstaben oder Ziffern) von der Kassette. CLOAD"PROG4" hat beispielsweise zur Folge, daß Ihr Programm mit dem Namen PROG4 von der Kassette aufgerufen wird. Wenn etwas schiefgeht, kann man das Laden mit CTRL/STOP unterbrechen, das Band zurückspulen und erneut mit dem Laden beginnen.
Wird der Dateiname weggelassen, so wird das erste auf der Kassette vorgefundene Programm eingelesen.

CLOAD?["<Dateiname>"]
Mit diesem Befehl kann ein einmal eingelesenes Programm mit dem ursprünglichen Programm verglichen werden. Ist alles in Ordnung, so erscheint "OK", bei einem Fehler erscheint "Verify error" (d.h. Fehler aufsuchen).

CLOSE [#][<Dateinummer>][,[#]<Dateinummer>...]
Dateien mit den angegebenen Nummern schließen. Wird keine Nummer genannt, so werden alle Dateien geschlossen.

CLS
Bildschirm löschen.

COLOR[<Vordergrundfarbe>][,[<Hintergrundfarbe>] [,<Randfarbe>]]
Hiermit werden die Farben der genannten Gebiete bestimmt. Die Farbe gibt man immer mit einer Zahl an:

Zahl	*Farbe*	*Zahl*	*Farbe*
0	transparent	8	rot
1	schwarz	9	hellrot
2	grün	10	dunkelgelb
3	hellgrün	11	hellgelb
4	dunkelblau	12	dunkelgrün
5	hellblau	13	magenta
6	dunkelrot	14	grau
7	himmelblau	15	weiß

Die Ausgangslage (Situation beim Einschalten des Computers) stimmt überein mit COLOR 15,4,7

Beispiele:
COLOR 2
Nur der Vordergrund wird grün.
COLOR ,,5
Nur der Hintergrund wird hellblau.
Bei grafischen Abbildungen kann man die Hintergrundfarbe nur dann ändern, wenn zuvor CLS ausgeführt wurde.

CONT
Mit diesem Befehl wird die Ausführung des Programmes wieder von der Stelle aus aufgenommen, an der es mit CTRL/STOP oder mit der STOP- oder END-Anweisung unterbrochen wurde.

CSAVE"<Dateiname>"[,<Baudgeschwindigkeit>]
Befehl zur Speicherung eines Programmes auf Kassette (siehe auch CLOAD). Für die Baudgeschwindigkeit gilt:

1 1200 Baud
2 2400 Baud
Wird diese Kennzeichnung ausgelassen, so wird von der Geschwindigkeit 1200 Baud ausgegangen.
Die Baudgeschwindigkeit kann auch mit SCREEN eingestellt werden.

DATA<n1>[,<n2>...]
Mit dieser Anweisung kann man eine Liste mit feststehenden Werten in ein Programm aufnehmen, die dann nacheinander mit einer READ-Anweisung eingefügt werden können. Diese Liste darf aus numerischen Werten wie aus Strings (zwischen Anführungszeichen (") bestehen, und diese müssen durch Kommas voneinander getrennt werden. READ ruft die Werte sequenziell auf. Man kann mit dem Aufrufen einer DATA-Liste wieder von vorn beginnen, indem man zuerst eine RESTORE-Anweisung vornimmt.

DEF FN <Funktionsname>[(<Argumentenliste>)]
=<Funktionsdefinierung>
Hiermit definiert man eigene Funktionen. Der Name, unter dem die Funktion aufgerufen werden muß, ist der <Funktionsname> mit zuvor eingegebenem FN. Bei String- Funktionen endet der Name auf $. Eventuell kann man der Funktion Argumente mitgeben, die dann in der <Funktionsdefinierung> gebraucht werden. Die Namen der Argumente dienen nur zur Definierung der Funktion und haben keinerlei Einfluß auf eventuelle Variablen in dem Hauptprogramm des gleichen Namens. Die Argumente müssen durch ein Komma getrennt werden.
In der <Funktionsdefinierung> dürfen auch Namen von Variablen vorkommen, die nicht als Argument in der <Argumentenliste> mitgegeben werden , sondern vor dem Aufruf der Funktion bereits im Hauptprogramm einen Wert erhalten haben. Die

<Funktionsdefinierung> darf höchstens eine Zeile lang sein.
Beispiel:

```
10 DEF FNAB(X,Y)=X∧2+Y*Z
20 Z=5
30 I=2:J=3
40 T=FNAB(I,J)
50 PRINT T
60 END

RUN
19
```

DEF <Typ><Buchstabe-Kennzeichnung>
Diese Anweisung sorgt dafür, daß alle Variablen mit genanntem <Buchstaben> vom angegebenen <Typ> sind. Die möglichen Typen sind:

INT	Integer
SGN	Real
DBL	Real mit doppelter Genauigkeit
STR	String

Ein Typen-Deklarierzeichen (also %, #, $) hat Vorrang gegenüber der DEF-Anweisung. Beispiele:

`10 DEFDBL A-E`	Alle Variablen, die mit A, B, C, D oder E beginnen, sind Variablen mit doppelter Genauigkeit.
`10 DEFSTR A`	Alle mit dem Buchstaben A beginnenden Variablen sind Stringvariablen.

DEF USR[<Zahl>]=<Speicheradresse>
Hiermit spezifiziert man die Beginnadresse einer Subroutine in Maschinensprache. <Zahl> muß zwischen 0 und 9 liegen und wird der Maschinensprachroutine als Name zugewiesen. Läßt man <Zahl> weg, so wird 0 angenommen. Auf Grund dieser <Zahl> und mit Hilfe der USR-Funktion kann man nun die betreffende Subroutine in Maschinensprache aufrufen. Beispiel:

```
10 DEF USR0=1000
20 X=URS0(9^2)
```

DELETE[<Zeilennummer>][-<Zeilennummer>]
Entfernt alle genannten Zeilen. Nach Erledigung dieses Befehls wird immer zum BASIC-Befehlsmodus zurückgekehrt.

Beispiele:

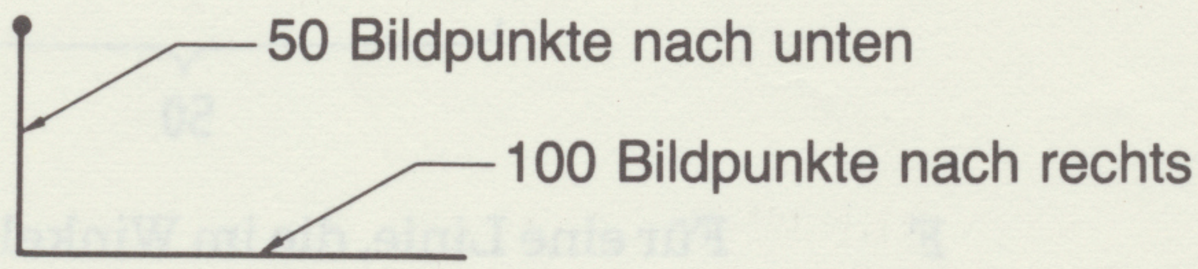
DELETE 10	Zeile 10 entfernen
DELETE 10–40	Zeilen 10-40 entfernen
DELETE –40	Alle Zeilen von 1-40 entfernen
DELETE 40–	Nicht gestattet
DELETE.	Letzte Zeile entfernen

DIM <Array-Name>(<Maximum Index>)[,<Array-Name>...]
Speicherraum für spezifizierte Array(s) schaffen und Array-Elemente auf Null initialisieren. Wenn ein Array angesprochen wird,das nicht zuvor durch eine DIM-Anweisung geschaffen worden ist, so wird 10 als Maximum Index angenommen. Der niedrigste Index ist immer 0. Mit ERASE können wir ein Array löschen.

DRAW <String>
Mit DRAW können wir mit einfachen Kodes allerlei gerade Linien angeben. Diese Kodes bilden immer die String.
Beispiel:

```
10 SCREEN 2
20 DRAW "50R100"
30 GOTO 30
```

Man erhält zwei aneinander anschließende Geraden. Die erste geht über 50 Bildpunkte nach unten ("down 50" oder abgekürzt D50) und die zweite 100 Punkte nach rechts R100). Das Ergebnis ist also wie folgt:



Für jeden Linienabschnitt können wir auch mit dem Buchstaben C und einem Farbkode eine Farbe angeben, z.B.:

```
DRAW "C8D50C10R100"
```

Die folgende Tabelle gibt eine Übersicht der Möglichkeiten:

Kode	*Bedeutung*
S	Maßstab. Nach S folgt eine Zahl. Der Maßstab entspricht dann der Zahl /4.
A	Nach A folgt 0, 1, 2 oder 3. Hiermit drehen wir das Koordinatensystem in 90° Schritten. Ohne A geht der Computer von A0 aus.
C	Farbangabe, siehe obiges Beispiel.

M Zum Zeichnen einer schrägen Linie. Nach M folgen zwei Zahlen, getrennt durch ein Komma, z.B.:

M30,50

In diesem Beispiel wird eine schräge Linie gezogen ab der letzten Position bis zu dem Punkt, den wir dadurch erhalten, daß wir zur x- Koordinate 30 und zur y-Koordinate 50 hinzuzählen. Die Werte x und y dürfen auch negativ sein.

U U bedeutet 'up', also 'nach oben'. Beispiel:

U30

gibt an, daß ab der letzten Position eine Linie über 30 Bildpunkte nach oben zu ziehen ist.

D D bedeutet 'down', also nach unten; zum Ziehen einer Linie nach unten.

R R bedeutet 'right' oder 'rechts'; zum Ziehen einer Linie nach rechts.

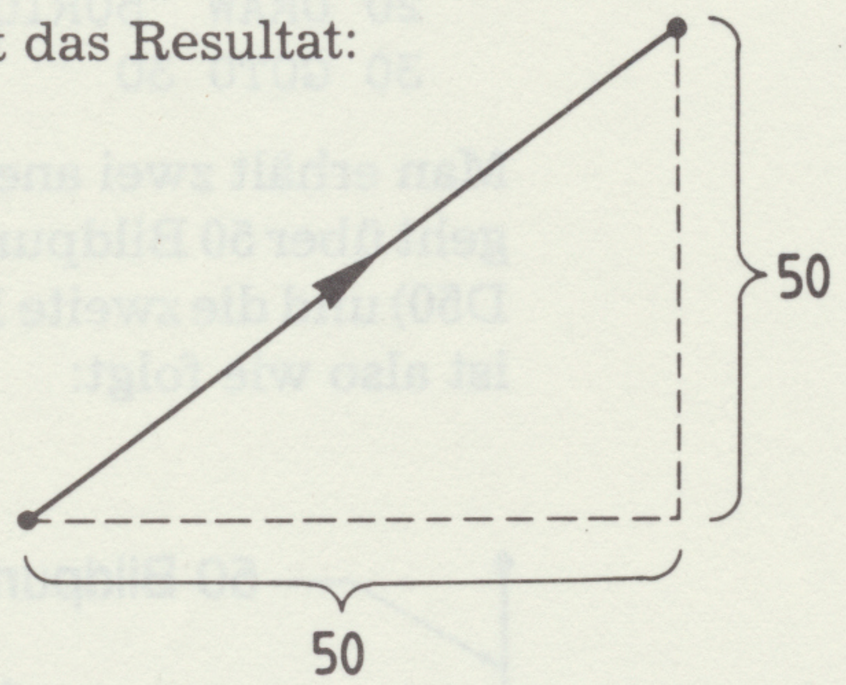
L L bedeutet 'left' oder 'links'; zum Ziehen einer Linie nach links.

E Für eine Linie, die im Winkel von 45° nach rechts oben gezogen wird.

Beispiel:

E50

erbringt das Resultat:



F Für eine Linie, die im Winkel von 45° nach rechts unten gezogen wird (siehe auch E).

G Für eine Linie, die im Winkel von 45° nach links oben gezogen wird (siehe auch E).

H Für eine Linie, die im Winkel von 45° nach links unten gezogen wird (siehe auch E).

Darauf achten, daß die Linien immer ab der zuletzt erreichten Position gezogen werden. Wir können eine bestimmte Ausgangsposition mit BMx,y erhalten, wobei x und y die Ausgangskoordinaten sind.

Beispiel:

```
DRAW "BM50,50D30"
```

ergibt ab Punkt (50,50) eine Linie nach unten über 30 Punkte.

Man kann einen zuletzt erreichten Punkt auch als Ausgangs-

punkt für eine neue Linie aufrechterhalten, indem dann vor den Kode ein N gesetzt wird. Man kann die String auch mit einer Stringvariablen angeben, beispielsweise:

```
....
50 B$="BM50,50D30"
60 DRAW B$
```

Man kann einen Teil einer String in einer DRAW-Anweisung auch mit einer Variablen wiedergeben. In den Teil der DRAW-Anweisung setzen wir dann ein X. Beispiel:

```
....
50 B$="BM50,50D30"
60 DRAW "R100XB$"
```

Zum Schluß kann man die Zahlen, die im DRAW-String vorkommen, auch mittels einer Variablen andeuten. Die Variable steht dann zwischen den Zeichen = und;. Beispiel

```
....
50 K=30
60 DRAW "B10,10R=K;"
```

END
Ende des Programmes. Diese Anweisung darf überall im Programm vorkommen. Ein END am Ende eines Programmes ist wahlfrei, d.h. nicht vorgeschrieben. Alle noch offenen Dateien werden damit geschlossen.

EOF (<Dateinummer>)
Bestimmt, ob das Ende einer Datei (File) während des Einlesens von Daten erreicht ist. Beispiel:

```
IF EOF(2) THEN CLOSE #2
```

ERASE <Array-Name>[,<Array-Name>...]
Mit ERASE können wir ein Array löschen, wodurch wieder Speicherraum frei wird (siehe auch DIM).
Beispiel:

```
10 DIM K(100), X$(50)
...
500 ERASE K,X$
```

ERR- und ERL-Variablen
Tritt in einem Programm ein Fehler auf, so enthält die Variable ERR die Fehlernummer des aufgetretenen Fehlers (siehe Anhang G, Fehlermeldungen) und enthält ERL die Zeilennummer, in der der Fehler aufgetreten ist.

ERR und ERL können in den vom Anwender geschriebenen Fehlerabhandlungsroutinen (siehe auch ON ERROR GOTO) angewandt werden, meist in IF...THEN-Konstruktionen. Beispiel: IF ERR=11 AND ERL=160 THEN ...usw.

ERROR <Fehlernummer>
Zeigt zu der genannten <Fehlernummer> gehörenden Fehlerkode auf dem Bildschirm. Die <Fehlernummer> muß größer als 0 und kleiner als 255 sein. Anhang G zeigt eine Übersicht der Fehlernummern. Beispiel: Eingeben ERROR 11. Auf dem Bildschirm erscheint /ØError.
Nicht alle Fehlernummern zwischen 0 und 255 werden von BASIC gebraucht. Die freien Nummern kann man verwenden, um eigene Fehlerberichte zu generieren.
Nachstehendes Beispiel zeigt, daß die Standard-Fehlermeldung nicht immer abgedruckt wird, sondern daß wir auch die selbst eingegebene Nachricht abdrucken lassen können. Beispiel:

```
 10 ON ERROR GOTO 400
 20 INPUT "X:-",A
 30 IF A>100 THEN ERROR 210
    ...
    ...
400 IF ERR=210 THEN PRINT "MAXIMAL 100!"
410 RESUME 20
```

FOR ... NEXT
Die vollständige Syntax ist:

FOR <Variable>=<n>TO<m>[STEP<k>]
Anweisung 1
Anweisung 2
...
...
NEXT[<Variable>][,<Variable>...]

wobei n, m und k numerische Ausdrücke sind.
Alle Anweisungen zwischen FOR und NEXT werden wiederholt ausgeführt. Dies geschieht so oft, bis der Zähler (=<Variable>) den Wert von m überschreitet. Der Ausgangswert des Zählers ist n, und der Zähler wird jedesmal nach Ausführung aller Anweisungen um k erhöht. Wird k nicht spezifiziert, so wird der Zähler immer um 1 erhöht.
FOR ... NEXT-Anweisungen dürfen ihrerseits andere FOR ... NEXT-Anweisungen enthalten.

GOSUB <Zeilennummer>RETURN<Zeilennummer>
Diese Anweisung ermöglicht einen Sprung zu einer Subroutine. <Zeilennummer> ist die Nummer der ersten Zeile der Subroutine. Ein RETURN muß irgendwo in der Subroutine vorkommen und sorgt dafür, daß zur hinter der GOSUB-Anweisung folgenden Anweisung zurückgekehrt wird. Subroutinen dürfen ihrerseits wieder auf andere Subroutinen verweisen.

GOTO <Zeilennummer>
Fortsetzung der Programmausführung bei der angegebenen <Zeilennummer>.

IF ... THEN
Die vollständige Syntax lautet:
IF <Boolescher Ausdruck>[<Boolescher Operator><Boolescher Ausdruck>...]THEN<Zeilennummer oder Anweisung(en)>[ELSE<Zeilennummer oder Anweisung(en)>]
Anmerkung: Mit 'Boolescher Ausdruck, wird gemeint 'logische Voraussetzung', die richtig oder falsch ist. In <Boolescher Ausdruck> dürfen als Bezugszeichen (Operator) stehen:

<	: kleiner als	<=	: kleiner als oder gleich
>	: größer als	>=	: größer als oder gleich
=	: gleich	<>	: ungleich

Werden nach THEN oder ELSE mehrere Anweisungen eingegeben, so müssen diese durch einen Doppelpunkt : voneinander getrennt werden.
Der Gesamtausdruck muß sich jedoch über eine Zeile erstrekken. In den Ausdrücken dürfen OR, AND, NOT, XOR, EQV oder IMP verwendet werden (siehe Anhang F).
Wenn der gesamte Ausdruck zwischen IF und THEN wahr ist, dann werden die Anweisungen nach THEN ausgeführt. Folgen nach THEN keine Anweisungen, sondern eine Zeilennummer, so wird mit der genannten Zeilennummer fortgesetzt.
Wenn der gesamte Ausdruck zwischen IF und THEN nicht wahr ist, dann werden die Anweisungen nach ELSE ausgeführt oder wird zur genannten Zeile gesprungen. Ist kein ELSE anwesend, wird mit der Zeile fortgesetzt, die unmittelbar hinter der IF...THEN-Anweisung folgt.
Die Anweisungen nach THEN oder ELSE dürfen wiederum eine IF...THEN-Anweisung enthalten, sofern diese ganz in einer Zeile steht. Also eine Anweisung

```
IF X>Y THEN PRINT "GROESSER" ELSE IF Y>X THEN PRINT
"KLEINER" ELSE PRINT "GLEICH"
```

ist gestattet. Jedes ELSE in solch einer Anweisung wird verbunden mit dem nächstgelegenen IF, für das noch kein zugehöriges ELSE gefunden worden ist.

INPUT["<Text>";]<Variable 1>[,<Variable 2>...]
Einlesen von Werten, die vom Anwender auf der Tastatur eingegeben werden. Auf dem Bildschirm erscheint immer ein Fragezeichen (?).
Wird <Text> mit eingegeben, so erscheint dieser Text auf dem Bildschirm vor dem Fragezeichen.
Die eingegebenen Werte werden den <Variablen> zugewiesen. Es müssen ebenso viele Werte, durch Komma voneinander getrennt, eingegeben werden, als Variablen in der INPUT-Anweisung vorkommen. Auch String-Variablen sind zulässig, aber in der eingegebenen String dürfen keine Kommas vorkommen. Wenn in den eingegebenen Werten ein Fehler auftritt, erscheint die Fehlermeldung ?Redo und wird die INPUT-Anweisung erneut ausgeführt.

INPUT #<Filenummer>,<Variable 1>[,<Variable 2>...]
Diese Anweisung ist vergleichbar mit INPUT, doch nun handelt es sich um die Werte in einer Datei. <Filenummer> ist die Nummer, die der betreffenden Datei mit OPEN zugewiesen ist. Die Werte in der Datei müssen in der gleichen Reihenfolge stehen, wie die Variablen in der INPUT-Anweisung. Während des Einlesens einer String wird das Ende der String bestimmt durch ein Komma, RETURN oder Line Feed. Ist das erste Zeichen der String jedoch ein Anführungszeichen ("), so wird ein zweites Anführungszeichen als Ende der String angenommen.

INTERVAL ON
INTERVAL OFF
INTERVAL STOP
Hiermit kann man angeben, ob eine Unterbrechung, die durch die eingebaute Uhr angegeben wird, eingeschaltet, ausgeschaltet oder eingehalten wird. Das Intervall muß in einer ON INTERVAL GOSUB-Anweisung angegeben werden.
Nach der Anweisung INTERVAL ON springt der Computer nach Verstreichen des Zeitintervalles zur Subroutine, die in ON INTERVAL GOSUB angegeben ist.

KEY <n>,"<Befehlsstring>"
Zuweisung einer <Befehlsstring> an eine Funktionstaste. <n> ist die Nummer der Funktionstaste (1 bis 10). Der <Befehlsstring> muß zwischen Anführungszeichen (") stehen und darf höchstens 15 Schriftzeichen haben oder wird durch eine String-Variable angegeben. Man kann ein RETURN in die <Befehlsstring> mit dem Zusatz +CHR$(13) aufnehmen. Beispielsweise eingeben:

```
KEY 1,"RUN"+CHR$(13)
```

Wird nun Funktionstaste 1 angeschlagen, so wird das Pro-

gramm sofort ausgeführt, weil der Befehl RUN mit einem RETURN beendet wurde.

KEY LIST
Vermittelt eine Übersicht der Befehlsstrings der 10 Funktionstasten.

KEY ON
KEY OFF
Bestimmung, ob die Bedeutung der Funktionstasten auf dem Bildschirm wiedergegeben werden oder nicht. Man kann eventuell eine der Tasten angeben, z.B.:

```
KEY (1) ON
```

KEY (<n>) ON
KEY (<n>) OFF
KEY (<n>) STOP
Hiermit kann man feststellen, ob eine der Funktionstasten gedrückt ist oder nicht. Mit ON schaltet man diesen Zustand ein und mit OFF wieder aus.
Mit STOP springt der Computer nicht unmittelbar gemäß ON KEY GOSUB in die Subroutine, sondern erst nach Eingabe von KEY (n) ON.

[LET]<Variable>=<Wert>
Diese Anweisung weist einer Variablen einen Wert zu. Der Ausdruck LET darf eventuell weggelassen werden.

LINE [[STEP](<X1,Y1>)]-[STEP]-(<X2,Y2>)[,Farbe][,BF oder B]
Zeichnet eine Linie zwischen (X1,Y1) und (X2,Y2). Mit STEP können wir angeben, daß das Koordinatensystem zum angewiesenen Punkt verlegt wird. Gibt man eine Diagonale an und schließt man diese Anweisung mit B, so wird ein Rechteck mit der entsprechenden Diagonalen gezeichnet. Mit BF wird das Rechteck gefärbt.

LINE INPUT ["<Text>";]<String-Variable>
Liest eine String ein, die auf der Tastatur eingegeben wird. Die String darf aus allen möglichen Schriftzeichen zusammengesetzt sein und eine beliebige Länge bis zu 255 Zeichen haben.

LINE INPUT #<Filenummer>,<String-Variable>
Diese Anweisung ist mit LINE INPUT vergleichbar, liest aber eine String aus einer Datei ein. <Filenummer> ist die mit OPEN zugewiesene Nummer der betreffenden Datei. Eine String wird bis zu einem RETURN insgesamt eingelesen.

LIST [<Zeilennummer>[-[<Zeilennummer>]]]
Dieser Befehl gibt die spezifizierten Zeilen auf dem Bildschirm wieder. Ohne Angabe von Zeilennummern wird das gesamte Programm wiedergegeben.
Beispiele:

LIST	Wiedergabe Gesamtprogramm
LIST 100	Wiedergabe nur von Zeile 100
LIST 100–150	Wiedergabe aller Zeilen von 100-150
LIST –100	Wiedergabe aller Zeilen von 1-100
LIST 100–	Wiedergabe aller Zeilen ab 100
LIST.	Wiedergabe nur der letzten behandelten Zeile

LLIST [<Zeilennummer>[-[<Zeilennummer>]]]
Wie LIST, nur werden jetzt die Zeilen auf dem Drucker ausgedruckt.

LOAD "<dev>:[Dateiname]"[,R]
Laden eines Programmes, das mit einem SAVE-Befehl auf die Kassette gespeichert wurde.
Beispiel:

```
LOAD "CAS:PROG1"
```

bedeutet: Von der Kassette das Programm einlesen, das unter dem Namen PROG1 gespeichert ist.

LOCATE [<X>][,[<Y>][,[Cursor an/aus>]]]
Versetzt den Cursor zur angegebenen Stelle. Der Cursor kann durch Angabe von 0 oder 1 ein- oder ausgeschaltet werden. X darf von 0-39 und Y von 0-23 variieren.

LPRINT [[USING<Druckformat>;]<Ausdruck>]
Wie PRINT oder PRINT USING, aber jetzt wird der Drucker angesteuert.

MAXFILE =<n>
Angabe der Höchstanzahl Files (Dateien), die zugleich geöffnet sein können.

MERGE "<dev>:[<Dateiname>]"
Fügt ein im externen Speicher vorhandenes Programm den im Speicher vorhandenen Programmen zu. Das Programm muß im ASCII-Format gespeichert sein.
Die Programmzeilen werden in ansteigender Reihenfolge der Zeilennummern gebracht. Haben zwei Zeilen die gleiche Zeilennummer, so wird die Zeile im Speicher durch die Zeile des eingelesenen Programmes ersetzt. Das neu entstandene Programm bleibt eingespeichert.

Beispiel:

```
MERGE "CAS:PROG1"
```

MID$(<String>,<erstes Zeichen>[,<Anzahl Zeichen>]= <String 2>

Mit diesem Befehl können wir <String 1> verändern. Wir geben dazu an, wieviele Zeichen und ab welchem Zeichen von String 1 die Veränderung zu geschehen hat durch Übernahme von String 2 in String 1.

Beispiel:

```
A$= "PHIPLIE"
MID$(A$,4,4)="LIPE"
```

Als Resultat wird A$ der String "PHILIPE" zugewiesen.

MOTOR ON
MOTOR OFF

Hiermit erfolgt Fernbedienung eines Kassettenrecorders. MOTOR ON hat den gleichen Effekt, wie die Betätigung der Taste "Play", und MOTOR OFF schaltet den Motor wieder aus.

NEW

Hiermit wird das eingespeicherte Programm wieder gelöscht.

ON ERROR GOTO <Zeilennummer>

Bei dieser Anweisung wird ein auftretender Fehler im Programm nicht auf dem Bildschirm, wie sonst üblich, gemeldet, sondern das Programm in der spezifizierten <Zeilennummer> fortgesetzt. Dadurch hat man die Möglichkeit, eigene Fehlerbehandlungsroutinen zu schreiben. <Zeilennummer> ist die erste Zeile der Fehlerbehandlungsroutine. In der Routine kann man die Variablen ERR und ERL anwenden.

Mit RESUME kehrt man von der Fehlerbehandlungsroutine wieder in das Hauptprogramm zurück. Die Anweisung kann wieder ungeschehen gemacht werden durch ON ERROR GOTO 0. Es ist empfehlenswert, in einer Fehlerbehandlungsroutine die Anweisung ON ERROR GOTO 0 zur Abwicklung unvorhergesehener Fehler auszuführen.

ON <Integer-Ausdruck>GOTO<Zeilennummer>[,<Zeilennummer>...]

Gestattet die Fortsetzung eines Programmes in einer Zeilennummer, die durch den Wert des <Integer-Ausdrucks> bestimmt wird. Wenn der <Integer-Ausdruck> beispielsweise den Wert 3 hat, wird das Programm mit der dritten Zeilennummer fortgesetzt, die nach GOTO spezifiziert ist. Der Wert des <Integer-Ausdrucks> darf nicht negativ, Null oder größer sein,

als die Anzahl der genannten Zeilennummern.
Hierzu ist anzumerken: wenn dieser Wert 0 ist oder <255, so setzt der Computer das Programm mit der folgenden ausführbaren Anweisung fort.

ON <Integer-Ausdruck>GOSUB<Zeilennummer>[,<Zeilennummer>...]
Funktioniert wie die Anweisung ON...GOTO, doch jetzt müssen die genannten Zeilennummern die erste Zeile einer Subroutine sein.

ON INTERVAL =<Zeit>GOSUB<Zeilennummer>
Hiermit wird angegeben, daß nach einer von INTERVAL genannten Wartezeit das Programm zu einer genannten Subroutine springen muß.
Beispiel:

```
...
ON INTERVAL=100 GOSUB 300
...
INTERVAL ON
...
```

Die Zahl nach dem Zeichen = gibt die Unterbrechungsdauer in 1/50 Sekunden an.

ON KEY GOSUB <Zeilennummer>[,<Zeilennummer>...]
Gibt an, zu welcher Subroutine gesprungen werden muß, sobald auf eine der Tasten F1 bis F10 gedrückt wird.

ON SPRITE GOSUB <Zeilennummer>[,<Zeilennummer>...]
Gibt an, in welche Subroutine gesprungen werden muß, wenn sich zwei Sprites überdecken. Zuvor muß jedoch die SPRITE ON-Anweisung gegeben werden.
Beispiel:

```
...
SPRITE ON
ON SPRITE GOSUB 300
...
```

ON STOP GOSUB <Zeilennummer>[,<Zeilennummer>...]
Gibt an, in welche Subroutine gesprungen werden muß, wenn das Programm mit CTRL/STOP unterbrochen wird.

```
...
STOP ON
ON STOP GOSUB 300
```

```
...
300 PRINT "CTRL/STOP GEDRUECKT!"
310 RETURN
```

ON STRIG GOSUB <Zeilennummer>[,<Zeilennummer>...]
Gibt an, in welche Subroutine gesprungen werden muß.
Bei Zeilennummer 1, 2, 3, 4 und 5 handelt es sich um:

Nr.	Feuerknopf oder Leertaste
1	Leertaste
2	Joystick 1, Feuerknopf 1
3	Joystick 2, Feuerknopf 1
4	Joystick 1, Feuerknopf 2
5	Joystick 2, Feuerknopf 2

Die betreffende Feuerknopf/Leertaste muß allerdings mit der Anweisung STRIG(<n>)ON aktiviert werden.
Beispiel:

```
... ...
100 ON STRIG GOSUB 300,400,500
110 STRIG(0) ON: REM LEERTASTE
120 STRIG(1) ON: REM JOYSTICK 1
130 STRIG(2) ON: REM JOYSTICK 2
```

OPEN "<dev>:[<Dateiname>]"FOR<Verarbeitung>AS [#]<Nummer>
Hiermit wird eine Datei (File) eröffnet.
Für <dev> kann folgendes eingesetzt werden:

CAS:	Kassettenrecorder
CRT:	Text-Bildschirm
GRP:	Grafik-Bildschirm
LPT:	Drucker

Als <Dateiname> kann ein Name mit höchstens 6 Zeichen genommen werden. Wenn als <Verarbeitung> OUTPUT genommen wird, handelt es sich um eine Datei, in die Daten eingegeben werden. Nimmt man INPUT, so handelt es sich um eine Datei, aus der Daten eingelesen werden.
Beispiel:

```
OPEN "CAS:PROG1" FOR OUTPUT AS#1
```

OUT <Anschlußnummer, Ausdruck>
Die durch <Ausdruck> gekennzeichneten Daten werden zum durch <Anschlußnummer> bezeichneten Anschluß gesteuert.

```
OUT &H80,3
```

PAINT [STEP](<X,Y>)[,[<Flächenfarbe>][,< Randfarbe>]]
Kennzeichnet die Einfärbung eines bestimmten Gebietes. Mit STEP wird das Koordinatensystem versetzt. Mit X und Y geben wir einen Punkt an. Eingefärbt wird die Fläche, in die dieser Punkt fällt. Bei nicht völlig geschlossener Randlinie wird der Gesamtbildschirm gefärbt.
Beim grafischen Bildschirm mit hoher Auflösung (SCREEN 2) gilt, daß der ganze Schirm eingefärbt wird, wenn die Flächenfarbe und die Farbe der zuvor gegebenen Randlinie nicht gleich sind.
Beispiel:

```
10 CLS
20 SCREEN
30 CIRCLE (100,100),50,10,,,1.4
40 PAINT (100,100),10
50 GOTO 50
```

Resultat: ein dunkelgelb gefärbter Kreis.

PLAY [<erster Kanal>][,[<zweiter Kanal>][,<dritter Kanal>]]
Anweisungen zum Hervorbringen von bestimmten Geräuschen. PLAY"CDE" läßt beispielsweise die Töne C, D und E erklingen. Zur Kennzeichnung einer Note (C, D, E, F, G, A, B und desgleichen C+ für Cis usw.) kann man Buchstaben und Zahlen mit einer besonderen Bedeutung einsetzen:

Kode *Bedeutung*

On Kennzeichnet die betreffende Oktave, z.B.:
PLAY"O5CDE": bedeutet: Spiele C, D und E von Oktave 5. Für 'n' gilt: 1 ≤n≤8. Ohne genauere Angabe geht der Computer von O4 aus.

Ln Angabe der Tondauer wie folgt:

Dauer	Kode
4fach	L1
2fach	L2
1fach	L4
1/2	L8
1/4	L16
1/8	L32
1/16	L64

Ohne genauere Angaben geht der Computer von L4 aus.

Nn Mit n zwischen 0 und 96: gibt einen Ton mit Nummer n an.

A bis G Angabe eines Tones. Gibt man beispielsweise A5 an, so stimmt dies überein mit O5A oder A in der fünften Oktave.

Rn Pausenzeichen in Etappen von 2. R4 bedeutet eine Viertelpause, R1 eine ganze Pause. Die entsprechenden Möglichkeiten sind also R1, R2, R4, R8, R16, R32 und R64.

Vn Lautstärkenangabe: n=15 ist die Maximallautstärke. Ohne ge-

nauere Angaben geht der Computer von V8 aus.

- Verlängert die Dauer der Note um die Hältte.

Sn Angabe der Klangfarbe (0 ≤n ≤15). Ohne genauere Angaben geht der Computer von S1 aus.

Tn Tempoangabe. Wert n bestimmt die Anzahl der Viertelnoten in einer Minute; n kann zwischen 32 und 255 liegen. Standardwert ist 120.

Mn Angabe des Maßes, in dem die Klangfarbe variiert. Ohne genauere Angabe geht der Computer von M255 aus (1 ≤M ≤65535).

POKE <Adresse, Daten>
Bringt Daten in angegebenes Speicherregister. Beispiel:

```
POKE 5000,100
```

PRESET [STEP](<x,y>)[,<Farbe>]
Setzt einen Punkt auf den grafischen Schirm (x,y) oder löscht ihn. Ist eine Farbe angegeben, so ist der Effekt von PRESET der gleiche wie von PSET. Ohne diese Kennzeichnung wird ein bereits angegebener Punkt gelöscht. Mit STEP kann man eventuell das Koordinatensystem versetzen.

PRINT [<Ausdruck>[, oder ;][<Ausdruck><, oder ;>...]]
Bildschirmwiedergabe des Wertes von Variablen, numerischen Ausdrücken oder Strings. Strings müssen zwischen Anführungsstriche (") gesetzt werden.
Die Bildschirmposition, in der die Daten eingesetzt werden müssen, wird durch das Trennungszeichen bestimmt. Ist dieses Trennungszeichen ein Semikolon (;) oder ein Freiraum, so werden die Daten unmittelbar hintereinander gesetzt. BASIC teilt eine Zeile in Zonen von 14 Spalten auf. Ist das Trennungszeichen zwischen zwei Angaben ein Komma (,), so wird die zweite Angabe in die nächste Zone gesetzt.
Schließt man eine PRINT-Anweisung mit einem Komma oder Semikolon ab, so werden die Daten bei der nächsten PRINT-Anweisung in die gleiche Zeile gesetzt, anderenfalls in die folgende Zeile. Den Ausdruck PRINT können wir auch durch ein Fragezeichen angeben, z.B.:

```
10 ? "ERGEBNIS"; A
```

PRINT USING "<Kodierung>";<Ausdruck>[;
<Ausdruck>...]
Diese Anweisung ist eine Erweiterung der PRINT- Anweisung, mit der man mittels der <Kodierung> selbst auch angeben kann, in welcher Form man die Daten wiedergeben möchte. Die <Ausdrücke> müssen durch ein Semikolon (;) getrennt werden.

Bei der Wiedergabe von Strings besteht die Wahl zwischen den folgenden drei <Kodierungen>:

! Nur das erste Schriftzeichen der String wird wiedergegeben.

\n Abstände Jetzt werden die ersten 2+n Schriftzeichen der String wiedergegeben.

& Die String wird vollständig wiedergegeben.

Bei der Wiedergabe der numerischen Werte besteht Wahl zwischen den folgenden speziellen Schriftzeichen:

Angabe der Positionenanzahl vor und hinter dem Komma. Zu große Zahlen werden abgerundet. Zu kleine Zahlen werden mit Nullen und Abständen passend gemacht. Ist die Zahl zu groß für die spezifizierte <Kodierung>, so erscheint vor der Zahl das Prozentzeichen (%).
Beispiele:

```
PRINT USING "##.##;1.2345
1.23
PRINT USING "##.##";99.996
%100.00
```

+ Ein Pluszeichen vor oder am Ende der <Kodierung> sorgt dafür, daß vor oder am Ende der Zahl ein Plus- oder Minuszeichen wiedergegeben wird.

– Ein Minuszeichen am Ende der <Kodierung> sorgt dafür, daß bei negativen Zahlen am Ende ein Minuszeichen wiedergegeben wird.

****** Ein doppelter Stern zu Beginn der <Kodierung> sorgt dafür, daß eventuelle Abstände vor der Zahl mit Sternchen-Zeichen angegeben werden. Diese beiden Sternchen zählen außerdem als zwei extra Positionen.

$$ Ein doppeltes $-Zeichen zu Beginn der <Kodierung> sorgt dafür, daß ein $-Zeichen vor der Zahl wiedergegeben wird. Die wissenschaftliche Schreibweise (siehe weiter) kann in Verbindung mit $$ nicht angewandt werden.

****$$** Eine Kombination der beiden oben genannten Effekte.

, Ein Komma links vom dezimalen Punkt sorgt dafür, daß alle drei Positionen ein Komma wiedergegeben wird. Beispiel:

```
PRINT USING "####,.##";1234.5
1,234.50
```

Ein Komma am Ende einer <Kodierung> wird normal wiedergegeben. Beispiel:

```
PRINT USING "####.##,";1234.5
1234.50,
```

^^^^ Durch vier Pfeile am Ende einer <Kodierung> wird die Zahl in wissenschaftlicher Schreibweise wiedergegeben. Beispiel:

```
PRINT USING "##.## ^^^^";123.45
1.23E+0.1
```

Text Text vor oder hinter der <Kodierung> wird, sofern von der <Kodierung> durch einen Abstand getrennt, normal wiedergegeben. Beispiel:

```
PRINT USING "DM ##.## MARK";12.34
DM 12.34 MARK
```

PRINT # und PRINT # USING

Die vollständige Syntax lautet:

```
PRINT #<Filenummer>,[USING"<Kodierung>";]<Aus-
druck>[;<Ausdruck>...]
```

Hiermit werden Daten zur Datei geschrieben. <Filenummer> ist die Nummer, die mit OPEN der Datei zugewiesen wurde. Für <Kodierung> gelten die gleichen Möglichkeiten, die bei der Anweisung PRINT USING beschrieben wurden. Die <Ausdrücke> werden hintereinander zur Datei geschrieben, und als Trennungszeichen darf nur das Semikolon (;) verwendet werden. Verwendet man Kommas, so werden auch die Zwischenräume, die zur Wiedergabe auf dem Bildschirm hinzugefügt werden, in die Datei geschrieben.
Auch Strings werden hintereinander in die Datei gegeben und sind daher nicht mehr als gesonderte Strings erkennbar. Ist beispielsweise A$=HANS und B$=KLAUS, so schreibt die Anweisung:

```
PRINT #1,A$;B$
```

und als Ergebnis geht HANSKLAUS in die Datei. Dieses Problem ist durch Einsatz von Trennungszeichen zu vermeiden. Beispiel:

```
PRINT #1,A$;",";B$
```

führt zu dem Ergebnis HANS,KLAUS.
Eine andere Möglichkeit ist es, die Strings in zwei Anweisungen in die Datei zu schreiben:

```
PRINT #1,A$
PRINT #1,B$
```

Da die PRINT-Anweisung nicht mit einem Semikolon abgeschlossen wird, erfolgt automatisch ein RETURN am Ende der String. RETURN fungiert nun als Trennungszeichen.
Enthält der String selbst ein Komma, so wird sie beim Zurücklesen als zwei Strings angesehen. Beispiel:
A$=HANS,KLAUS und die Anweisung

```
PRINT #1,A$
```

ergibt in der Datei HANS,KLAUS und erbringt nach dem Einlesen mit

```
INPUT #1,A$,B$
```

als Resultat A$=HANS und B$=KLAUS. Dieses Problem ist zu vermeiden, indem man die String zwischen Anführungsstrichen (") in die Datei setzt (siehe auch INPUT-Anweisung). Die Anweisung wird damit:

```
PRINT #1,CHR$(34);A$;CHR$(34)
```

34 ist der ASCII-Code für Anführungsstriche (").

PSET [STEP](<x,y>)[,<Farbe>]
Siehe PRESET; jetzt handelt es sich um die Wiedergabe eines Punktes.
Beispiel:

```
10 CLS
20 SCREEN 2
30 PSET (100,50)
40 GOTO 40
```

PUT SPRITE <Flächennummer>[,[STEP](x,y)][,<Farbe>]], <Sprite-Nummer>]
Setzt Sprite mit der angegebenen Farbe und Nummer in Position (x,y).
Die erste Nummer ist die Prioritätsnummer des Sprite. Beispiel:

```
PUT SPRITE 0, (100,50),1,3
```

Mit STEP kann man eventuell das Koordinatensystem versetzen. Für ein 8x8 Sprite kann man als <Sprite- Nummer> eine Zahl zwischen 0-255 wählen, für ein 16x16 Sprite eine Zahl zwischen 0-63.
Wenn 2 Sprites sich auf dem Bildschirm in gleicher Position befinden, dann ist das Sprite mit der niedrigsten Prioritätsnummer sichtbar.

READ <Variable>[,<Variable>...]
READ wird immer in Verbindung mit DATA angewandt und weist die durch DATA spezifizierten Werte den genannten <Variablen> zu. Siehe ferner DATA.

REM <Kommentar>
Hiermit kann einem Programm ein Kommentar hinzugegeben werden. Alle Informationen hinter REM werden von BASIC negiert. REM kann durch das Symbol ' abgekürzt werden.

RENUM [<neue Zeilennummer>][,[<alte Zeilennummer>][,<Schrittgröße>]]]
Zur Neunumerierung von Zeilen. <neue Zeilennummer> ist die Ausgangsnummer der ersten Zeile. Ohne besondere Angaben wird mit 10 begonnen.
<alte Zeilennummer> gibt an, ab welcher Zeile die Neunumerierung beginnen soll. Wird <alte Zeilennummer> nicht angegeben, so wird mit der ersten Zeile begonnen.
<Schrittgröße> ist die Zahl, um die die Zeilennummer jeweils erhöht werden muß. Normalerweise ist das 10.

RESTORE [<Zeilennummer>]
Hiermit können durch DATA spezifizierte Listen wieder von Beginn an mit READ gelesen werden. Ferner siehe DATA. Man kann eventuell mit <Zeilennummer> zu einer betreffenden Zeile mit DATA springen.

RESUME <Zeilennummer>
RESUME wird nur in Fehlerbehandlungsroutinen gebraucht und hat die gleiche Funktion wie RETURN in Subroutinen. <Zeilennummer> gibt die Zeile an, von der aus die Programmausführung fortgesetzt werden muß. (Siehe ON ERROR GOTO ...)

RUN [<Zeilennummer>]
Beginn einer Programmausführung ab <Zeilennummer>. Wird keine <Zeilennummer> angegeben, so beginnt die Ausführung bei der ersten Zeile.

RUN "CAS:<Name>"[,R]
Hiermit kann man ein in die Kassette unter <Name> gespeichertes Programm starten. Mit ,R wird angegeben, daß eventuell geöffnete Dateien nicht geschlossen werden.

SAVE "<dev>:[<Dateiname>]"
Zur Speicherung einer Datei (File).

<dev-Name>	Bedeutung
CAS	Kassette (in ASCII-Codes)
CRT	Text-Bildschirm
GRP	Grafischer Bildschirm
LPT	Drucker

Beispiel:

```
SAVE "CAS:PROG1"
```

bedeutet: BASIC-Programm unter dem Namen PROG1 auf Kassette speichern.

SCREEN [<Modus>][,[<Sprite-Format>][,[<i>][,[<s>][,[<a>]]]]]

Mit der SCREEN-Anweisung wird dem Computer angegeben, wie der Bildschirm gebraucht werden soll. Normalerweise gebraucht man SCREEN 2 um anzugeben, daß der grafische Bildschirm mit hoher Auflösung angewandt werden soll. Ohne genauere Angabe geht der Computer von der Anweisung SCREEN 0,0,1,1,0 aus.

Es gelten folgende Regeln:

<Modus>	Bedeutung
0	Text-Bildschirm: 40x24 (Zeilen)
1	Text-Bildschirm: 32x24 (Zeilen)
2	grafischer Bildschirm (hohe Auflösung = 256 x 192 Raster)
3	grafischer Bildschirm (niedrige Auflösung = 64 x 48 Rasterblöcke)
<Spriteformat>	
0	8x8, ohne Vergrößerung
1	8x8, mit Vergrößerung
2	16x16, ohne Vergrößerung
3	16x16, mit Vergrößerung
<i>	(ohne oder mit Geräusch bei Tastendruck)
0	kein Piepton bei Tastendruck
andere Zahl	Piepton bei Tastendruck
<s>	(Geschwindigkeit Eingabe/Ausgabe von Kassette)
1	1200 Baud
2	2400 Baud
<a>	(Art Drucker)
0	MSX-Typ
1	Andere

SOUND <Registername, Zahl>

Anweisung zur Generierung eines bestimmten Tones. Es gelten folgende Regeln:

Register	Bereich Zahl	Bedeutung
0	0-255	Frequenz Kanal A
1	0-15	Frequenz Kanal A
2	0-255	Frequenz Kanal B
3	0-15	Frequenz Kanal B
4	0-255	Frequenz Kanal C
5	0-15	Frequenz Kanal C
6	0-31	Frequenz Rauschen
7	0-63	Kanalwahl: Ton und Rauschen
8	0-15	Volumen Kanal A
9	0-15	Volumen Kanal B
10	0-15	Volumen Kanal C
11	0-255	Frequenz Mustervariation
12	0-255	Frequenz Mustervariation
13	0-14	Musterwahl

SPRITE ON
SPRITE OFF
SPRITE STOP
Mit ON, OFF und STOP wird angegeben, ob der Computer auf das Auftreten 'zusammenstoßender' Sprites achtet. Siehe ON SPRITE GOSUB. Mit SPRITE STOP wird angegeben, daß der Unterbrechungszustand einzuhalten ist. In diesem Fall wird erst zu der mit ON SPRITE GOSUB angegebenen Subroutine gesprungen, wenn wieder SPRITE ON abgetroffen wird.

STOP
Unterbricht die Programmausführung. Wir können das Programm dann wieder, mit dem CONT-Befehl fortsetzen.

STOP ON
STOP OFF
STOP STOP
Regelt den Modus bei einer durch CTRL/STOP verursachten Unterbrechung (siehe ON STOP GOSUB). Bei STOP STOP wird der Unterbrechungszustand eingehalten, d.h. erst nach einem Vorfinden von STOP ON wird in die durch ON STOP GOSUB vorgefundene Subroutine gesprungen.

STRIG (<X>)ON
STRIG (<X>)OFF
STRIG (<X>)STOP
Regelt den Modus bei einer durch Joystick oder Leertaste verursachten Unterbrechung (siehe ON STRIG GOSUB). Bei STRIG STOP wird der Unterbrechungszustand eingehalten, d.h. erst dann in die von ON STRIG GOSUB angegebene Subroutine gesprungen, wenn die STRIG ON- Anweisung wieder vorgefunden wird.

SWAP <Variable>,<Variable>
Austausch der Werte von zwei Variablen. Beispiel:

```
SWAP A,B
```

Beide Variablen müssen vom gleichen Typ sein.

TROFF
TRON
Durch den Befehl TRON wird der Computer in einen Zustand geschaltet, in dem während der Durchführung des Programmes zugleich die Zeilennummer der Zeile abgebildet wird, in welcher der Computer tätig ist.
Hierdurch wird die Fehlersuche erleichtert. Mit TROFF wird dieser Zustand wieder ausgeschaltet.

VPOKE <Adresse,Daten>
Wie POKE, aber jetzt bezieht sich <Adresse> auf eine Adresse im Video-Teil des RAM-Speichers. Die Adresse darf variieren zwischen 0-16383. Die Daten dürfen variieren zwischen 0-255.

WAIT <Anschlußnummer, Ausdruck 1>[,<Ausdruck 2>]
Bezieht sich auf das Einlesen von Daten über den I/O-Anschluß mit angegebener Anschlußnummer.
Der eingelesene Wert wird mit <Ausdruck 1> über die sogenannte exklusive OR-Bedingung kombiniert. Das Ergebnis wird über eine AND-Operation mit <Ausdruck 2> kombiniert. Ist das Resultat hiervon 0, so wird das Einlesen fortgesetzt, wenn nicht, setzt der Computer das Programm mit der folgenden Anweisung fort. Wird <Ausdruck 2> weggelassen, dann wird vom Wert 0 ausgegangen.

WIDTH (<Anzahl Spalten>)
Einstellung der Spaltenzahl des Text-Bildschirms. Ohne Kennzeichnung wird der Schirm in 40 Spalten aufgegliedert.
Beispiel:

```
WIDTH 20
```

Dies bedeutet, daß pro Zeile höchstens 20 Zeichen abgebildet werden können. Im Text-Modus 1 kann der Wert zwischen 1-40 variieren; im Text-Modus 2 zwischen 1-32.

Anhang E Übersicht der BASIC-Funktionen und Systemvariablen

Dieser Anhang vermittelt die vollständige Syntax aller BASIC-Funktionen. Hierbei werden zwei Notierungen mit folgender Bedeutung verwendet:

<X> Eine numerische Variable oder ein vom Anwender selbst eingesetzter Ausdruck;

<X$> Eine String-Variable oder ein vom Anwender selbst eingesetzter Ausdruck.

ABS(<X>)
Gibt absoluten Wert von <X>.

ASC(<X$>)
Gibt den ASCII-Code des ersten Schriftzeichens von <X$>.

ATN(<X>)
Gibt den Arctangens von <X> im Bogenmaß. Das Resultat liegt zwischen -π/2 und π/2.

BIN$(<X>)
Verwandelt die angegebene Zahl in eine binäre Notierung.

CDBL(<X>)
Verwandelt <X> in eine Zahl mit doppelter Genauigkeit.

CHR$(<ASCII-Code>)
Gibt das Schriftzeichen des betreffenden <ASCII-Code> an.

CINT(<X>)
Rundet den Wert <X> auf eine ganze Zahl (Integer) ab. Der Wert muß zwischen -32768 und 32767 liegen.

COS(<X>)
Vermittelt den Cosinus von <X>.<X> in Bogenmaß.

CSGN(<X>)
Verwandelt den Wert von <X> in eine Zahl mit einfacher Genauigkeit.

CSRLIN
Gibt die y-Koordinate der Position an, auf der sich der Cursor befindet.

EOF (<Dateinummer>)
Gibt an, ob das Ende der betreffenden Datei erreicht ist.

ERL
Siehe Anhang D.

ERR
Siehe Anhang D.

EXP(<X>)
Berechnet den Exponenten von <X>.

FIX(<X>)
Berechnet den ganzen Teil von <X>.

FRE(0) oder FRE(<" ">)
Ist das Argument eine 0dann gibt FRE die Größe des noch nicht benutzten Speicherraumes in Bytes an. Ist das Argument eine String-Variable, so gibt FRE die Anzahl freier Bytes im Speicher an, die für String-Variablen reserviert sind. Letztgenannter Raum kann mit CLEAR angepaßt werden.

HEX$(<X>)
Gibt eine String, die den Hexadezimalwert von <X> darstellt. <X> wird zuerst auf eine ganze Zahl abgerundet. <X> muß zwischen -32768 und 65535 liegen.

INKEY$
Diese Funktion wird in der Form <String-Variable>=INKEY$ gebraucht. Die Funktion kontrolliert, ob eine Taste der Tastatur gedrückt wird. Wenn ja, so wird das betreffende Zeichen der <String-Variablen> zugewiesen; wenn nicht, so enthält <String-Variable> eine leere String (sogenannten Null-String).
Beispiel:

```
10 PRINT "EIN H TIPPEN";
20 A$=INKEY$
30 IF A$<>"H" THEN 20
40 PRINT A$
50 END
```

INP(<Anschlußnummern>)
Angabe der Anschlußnummer für die Dateneingabe. Für die Datenausgabe wird OUT verwendet (siehe OUT).
Nachstehende Tabelle gibt einige Beispiele:

Art	*Nummer*	*Anwendung*
RS232-C	&H80	Datenein- und -ausgabe
	&H81	Modus (beim Einlesen) und Status (beim Auslesen)
Drucker	&H90	Signalanschlüsse (Einlesen)
		Status (Auslesen)
	&H91	Einlesen von Daten

INPUT$(<X>)/INPUT$(<X>,[#]<Y>)
Diese Funktion wird in der Form <String-Variable>=INPUT$(<X>) angewandt. Die Funktion liest eine String von der Tastatur in <X> Schriftzeichen ein. Statt der Tastatur kann auch Datei <Y> eingegeben werden. Alle Schriftzeichen werden akzeptiert, ausgenommen CTRL/C.

INSTR([I,]<X$>,<Y$>)
Kontrolliert, ob <Y$> in <X$> vorkommt und vermittelt die Position in <X$>. Kommt <Y$> nicht in <X$> vor, den gibt INSTR den Wert 0. INSTR sucht ab Beginn von <X$>, sofern man nicht mit [I,] eine andere Ausgangsposition angibt. Ist <Y$> eine leere String, so gibt INSTR den Wert 1.

INT(<X>)
Diese Funktion gibt die größte ganze Zahl an, die kleiner oder gleich <X> ist.

LEFT$(<X$>,<I>)
Gibt eine String, die aus den ersten <I> Schriftzeichen von <X$> besteht. <I> muß zwischen 0-255 liegen.

LEN(<X$>)
Angabe der Anzahl von Schriftzeichen, aus denen <X$> besteht. Gezählt werden auch Zwischenräume.

LOG(<X>)
Gibt den natürlichen Logarithmus von <X> an. <X> muß größer sein als 0.

LPOS(<X>)
X kann hier eine beliebige Zahl sein. LPOS gibt die Position des Druckers im Pufferspeicher des Druckers an.

MID$(<X$>,<I>[<,J>])
Gibt eine String, die ein Teil von <X$> ist. Die String beginnt in Position <I> und ist <J> Schriftzeichen lang. Ohne Angabe von <J> werden alle Schriftzeichen ab <I> genommen.

OCT$(<X>)
Gibt eine String mit dem Oktalwert von <X>. Zuerst wird <X> auf eine ganze Zahl abgerundet.

PAD(<X>)
Informiert über den Berührungsplatz gemäß nachstehender Tabelle:

Wert X	*Bedeutung des Funktionswertes*
0 oder 4	bestimmt ob Stelle berührt wird; 0 (keine Berührung) oder –1 (Berührung)
1 oder 5	X-Koordinate der berührten Stelle
2 oder 6	Y-Koordinate der berührten Stelle
3 oder 7	bestimmt, ob Schalter betätigt ist (0=nicht und –1=ja)

PDL(<X>)
Angabe des 'Paddle'-Standes (1-12). Wert variiert zwischen 0-255.
Für <X>=1,3,5,7,9oder 11 geht der Computer davon aus, daß der Joystick über Verbindung 1 angeschlossen ist. Bei <X>=2,4,6,8,10 oder 12 wird vom Anschluß 2 ausgegangen.

PEEK(<X>)
Inhaltsangabe der Speicheradresse X. In Form einer Dezimalzahl muß X zwischen -32768 und 65536 liegen. Ist X negativ, so wird die sogenannte 2-Ergänzung angewandt: PEEK (–1)=PEEK(65536 – 1)

PLAY(<X>)
Informiert über die gebrauchten Tonkanäle: 0 = alle Kanäle, 1 = Kanal 1, 2 = Kanal 2, 3 = Kanal 3.

POINT(<X>,<Y>)
<X>: Position in der Zeile (0-255)
<Y>: Zeile (0-191)
Vermittelt eine Zahl, die die Farbe des Punktes auf dem Bildschirm mit den Koordinaten (<X>,<Y>) angibt. Die möglichen Zahlen sind:

0	weiß (also kein sichtbarer Punkt)
1	schwarz
2 und 3	Farben bei Verwendung eines Farbfernsehers

POS(0)
Angabe der Cursor-Position in der Zeile. Die am weitesten links liegende Position ist Position 0. 0 ist nur ein Scheinargument und hat keine weitere Bedeutung.

RIGHT$(<X$>,<I>)
Gibt eine String, bestehend aus den letzten <I> Schriftzeichen von <X$>.

RND (<X>)
Gibt eine Zufallszahl zwischen 0 und 1. Welche Serie von Zufallszahlen generiert wird, hängt von der Anfangszahl ab.

<X>= negativ	Es wird von vorn begonnen mit der Serie Zufallszahlen.
<X>= positiv	Vermittelt die folgende Zufallszahl in der Serie.
<X>= 0	Gibt die letzte Zufallszahl nochmals.

Mit RND (–TIME) bekommen wir tatsächlich jedesmal andere Zahlen. (Vergleiche mit RANDOMIZE anderer BASIC-Versionen.)

SGN(<X>)
Ergibt den Wert –1, 0 oder 1, je nachdem ob X negativ, 0 oder positiv ist.

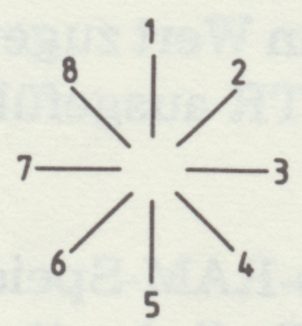
SIN(<X>)
Ergibt den Sinus von <X>. <X> muß im Bogenmaß angegeben werden.

SPACE$(<X>)
Ergibt eine String, bestehend aus <X> Zwischenräumen. <X> wird auf eine ganze Zahl abgerundet und muß zwischen 0 und 255 liegen.

SPC(<X>)
Gibt auf dem Bildschirm <X> Zwischenräume wieder. SPC darf nur innerhalb einer PRINT- oder LPRINT-Anweisung angewandt werden. <X> muß zwischen 0-255 liegen. Beispiel:

```
PRINT "FUER":SPC(10);"BILD"
FUER          BILD
```

SQR(<X>)
Erechnet die Wurzel aus <X>. <X> muß größer oder gleich 0 sein.



STICK(<X>)
Gibt die Stellung des Joysticks (1 oder 2) an. Setzt man für X den Wert 0 ein, so werden die Cursor-Tasten in Betracht gezogen.
Die Werte entsprechen nebenstehenden Richtungen:

STRING$(<I>,<J>) oder STRING$(<I>,<X$>)
Vermittelt eine String aus <I> gleichen Schriftzeichen, die als

ASCII-Code <J> haben oder mit dem ersten Schriftzeichen von <X$> übereinstimmen.

STR$(<X>)
Gibt eine String, die den Dezimalwert von <X> darstellt.

STRIG(<X>)
Gibt an, ob der Feuerknopf des Joysticks gedrückt wird (–1=ja, 0=nein)

<X>	*Bedeutung*
0	Leertaste Tastatur
1 oder 3	Joystick 1
2 oder 4	Joystick 2

TAB(<X>)
Bringt den Cursor in Zeilenposition <X>. Ist der Cursor bereits über Position <X> hinaus, so geschieht nichts.
<X> muß zwischen 0-255 liegen. 0 ist die Position links außen.
TAB kann nur in einer PRINT- oder LPRINT- Anweisung angewandt werden.
Beispiel:

```
PRINT "FUER";TAB(10);"BILD"
FUER        BILD
```

TAN(<X>)
Gibt Tangens von <X>. <X> muß im Bogenmaß angegeben werden.

USR[<n>](<X>)
Hiermit weist man ein Maschinensprache-Programm an. <n> ist die dem Maschinensprache-Programm mit DEF USR zugewiesene Nummer. Wird <n> weggelassen, so wird von USR0 ausgegangen.
<X> ist ein Argument, das dem Maschinensprache-Programm übermittelt wird.

VAL(<X$>)
Ergibt den numerischen Wert von <X$>. Wenn das erste Zeichen von <X$> kein +, –, & oder eine Ziffer ist, wird VAL gleich 0.

VARPTR(<Variable>)
Gibt die Adresse des ersten Bytes des Wertes, der zur <Variablen> gehört. Der <Variablen> muß ein Wert zugewiesen worden sein, bevor die Anweisung VARPTR ausgeführt wird.

VPEEK(<X>)
Hier gibt <X> eine Adresse im Video-RAM-Speicher an. Mit dieser Funktion kann der Inhalt dieser Stelle bestimmt werden.
<X> muß zwischen 0-16383 liegen.

Spezielle System-Variablen
MSX-BASIC hat noch eine Anzahl sogenannter System-Variablen. Diese Variablen haben einen reservierten Namen. Das System weist ihnen Werte zu, aber der Anwender kann diese auf Wunsch 'abschreiben'.

BASE(<X>)
Enthält die erste Adresse des sogenannten Video Display Processors (VDP).
<X> Bedeutung (Text-Modus = TM, Grafik-Modus = GM)

0 Name-Tabelle TM 1	12 Muster-Tabelle GM 1
2 Muster-Tabelle TM 1	13 Sprite-Attribut-Tabelle GM 1
5 Name-Tabelle TM 2	14 Sprite-Muster-Tabelle GM 1
6 Farbe-Tabelle TM 2	15 Name-Tabelle GM 2
7 Muster-Tabelle TM 2	16 Farbe-Tabelle GM 2
8 Sprite-Attribut Tabelle TM 2	17 Muster-Tabelle GM 2
	18 Sprite-Attribut-Tabelle GM 2
9 Sprite-Muster-Tabelle TM 2	19 Sprite-Muster-Tabelle GM 2
10 Name-Tabelle GM 1	
11 Farbe-Tabelle GM 1	

TIME
Der MSX-Computer hat eine interne Uhr, die die Zeit in TIME speichert. Wir können diese Variable vor allem gebrauchen, um wirklich willkürliche Zahlen über eine RND-Funktion zu bekommen (siehe dort).

SPRITE$
Hierbei handelt es sich um eine System-Variable, die immer in folgender Form angewendet wird: SPRITE$(<Nummer>)=(<String-Ausdruck>). Die <Nummer> gibt die sogenannte Sprite-Nummer an. Je nach der über die SCREEN-Anweisung gegebenen Definition des Sprite-Formats kann <Nummer> maximal 63 bzw. 255 sein. Mit dem <String- Ausdruck> geben wir die Sprite-Form an. Im allgemeinen tun wir das mit einer Anzahl von CHR$-Funktionen, z.B.:

SPRITE$(1) = `CHR$(&H18)+CHR$(&H3)+CHR$(&HFF)+CHR$(&H99)+`
`CHR$(&H99)+CHR$(&H99)+CHR$(&HFF)+CHR$(&HC3)+`
`CHR$(&HFF)`
Für ein 8x8 Sprite benötigt man so 8 CHR$-Funktionen.

VDP
Variable, die sich auf den sogenannten Video Display Processor bezieht.

Anhang F Besondere Zeichen und Ausdrücke

Bearbeitungszeichen

Nachstehende Tabelle vermittelt eine Übersicht der Zeichen für mathematische Bearbeitungen. Die letzte Ziffer gibt den Prioritätswert an: Ausdrücke werden diesen Prioritätswerten gemäß ausgearbeitet (von der niedrigsten bis zur höchsten Ziffer). Bei gleichem Prioritätswert wird der Ausdruck von links nach rechts verarbeitet.

Zeichen	*Bedeutung*	*Beispiel*	*Priorität*
+	Addieren	3+5	6
–	Subtrahieren	5–3	6
*	Multiplizieren	5x3	3
/	Dividieren	5/3	3
^	Potenzieren	2^3	1
–	Verändert Zeichen	–3	4
MOD	Rest bei Integerteilung	12 MOD 3	5

Zeichen in logischen Ausdrücken

Folgende Symbole dürfen in Ausdrücken vorkommen, wobei festgestellt wird, ob sie wahr sind oder nicht (z.B. nach IF)

Zeichen	*Bedeutung*	*Beispiel*
=	ist gleich	IF A= B THEN...
<	kleiner als	IF A< B THEN...
>	größer als	IF A> B THEN...
<> oder ><	ungleich	IF A<> B THEN...
<= oder >=	kleiner oder gleich	IF A<= B THEN...
>= oder =>	größer oder gleich	IF A>= B THEN...

Formeln zur Verbindung logischer Ausdrücke

Ausdrücke wie A=B und C<>D kann man mit spezifischen Formeln verbinden.
Folgende Tabelle zeigt die verschiedenen Formeln mit den sogenannten Wahrheitstabellen. Hierin gibt 1 an, daß die Erklärung 'wahr' ist (stimmt) und 0, daß die Erklärung 'unwahr' ist (nicht stimmt, falsch). So sieht man beispielsweise bei AND

A1 **AND** U2

hat 1 wie U1 und U2 gleich 1 zu sein. Mit U1 und U2 werden dann die Erklärungen A=B und C<>D gemeint. Die Tabelle gibt in diesem Fall an, daß die vollständige Erklärung stimmt (1), wenn U1 und U2 auch stimmen (1).

Ausdruck			*Wahrheitstabelle*
NOT	U1		NOT U1
(Verneinung)	1		0
	0		1
AND	U1	U2	U1 AND U2
(logisches Produkt)	1	1	1
	1	0	0
	0	1	0
	0	0	0
OR	U1	U2	U1 OR U2
(logische Summe)	1	1	1
	1	0	1
	0	1	1
	0	0	0
XOR	U1	U2	U1 XOR U2
(exklusives OR)	1	1	0
	1	0	1
	0	1	1
	0	0	0
EQV	U1	U2	U1 EQV U2
	1	1	1
	1	0	0
	0	1	0
	0	0	1
IMP	U1	U2	U1 IMP U2
(logische Implikation)	1	1	1
	1	0	0
	0	1	1
	0	0	1

Bearbeitungszeichen bei Strings

Bei Strings darf nur das Zeichen + zur Verbindung von zwei Strings verwendet werden. Beispiel:
"AB"+"CD" wird "ABCD".

Anhang G Fehlermeldungen

Bad allocation table (60)
Die Diskette ist nicht initialisiert (nicht oder unrichtig formatiert).

Bad drive name (62)
Es wird ein Diskettenantrieb angewiesen, der nicht vorhanden ist.

Bad file mode (61)
Man gebraucht PUT, GET oder EOF mit einer sequentiellen File oder versucht mit LOAD eine RANDOM-Datei zu laden. Diese Fehlermeldung kann auch gegeben werden, wenn die OPEN-Anweisung bei einer nicht einwandfrei definierten Datei gebraucht wird.

Bad file name (56)
Der Name der Datei (File) ist unrichtig. Der Name vor dem 'Device' in einem OPEN-, SAVE- oder LOAD-Befehl ist unrichtig.

Bad file number (52)
Die File-Nummer ist größer als erlaubt, oder man nennt eine noch nicht 'eröffnete' File.

Can't CONTINUE (17)
Man versucht einen CONT-Befehl zu geben, während das Programm nicht durch einen Fehler oder eine STOP-Anweisung unterbrochen worden ist. Ergibt sich auch, wenn wir das Programm ändern und sofort danach CONT probieren.

Device I/O error (19)
Das Laden eines Programmes oder Datei geht daneben. Ist der Kassettenrecorder richtig eingestellt? Ist der richtige I/O-Anschluß angegeben?

Direct statement in file (57)
In einem Programm, das geladen werden soll, kommt eine Anweisung ohne Zeilennummer vor, oder die zu ladende Datei ist kein Programm.

Disk full (66)
Die Gesamtkapazität der Diskette ist benutzt.

Disk I/O error (69)
Bei der Eingabe- oder Ausgabehandlung wird ein Fehler festgestellt. Dies ist ein sogenannter 'fatal error' (Kapitalfehler), d.h. das MSX Disk BASIC-System muß von Anwender erneut gestartet werden.

Disk offline (70)
Keine Diskette im angegebenen Antrieb.

Disk write protected (68)
Man versucht, etwas auf einer 'write protected', d.h. einer gegen Speicherung geschützten Diskette zu speichern.

Division by zero (11)
Teilen durch 0 nicht gestattet. Dieser Fehler kann auftreten, wenn man durch eine Variable dividiert, der zuvor noch kein Wert zugewiesen worden ist.

File already exists (65)
Der neue Name einer Datei (durch NAME-Befehl) ist ein bereits vorhandener Name einer Datei auf der Diskette.

File already open (54)
Man versucht eine Datei zu öffnen, die bereits geöffnet ist.

File not found (53)
Ein LOAD-, KILL-Befehl oder eine OPEN-Anweisung, die sich auf eine Datei bezieht, die nicht auf der Diskette ist.

File not OPEN (59)
Es wird auf eine Datei hingewiesen, die noch nicht mit einer OPEN-Anweisung geöffnet ist.

File still open (64)
Die Datei ist noch nicht geschlossen.

File already open (54)
Man kann keine Datei öffnen, die bereits geöffnet ist.

Illegal direct (12)
Man versucht die Ausführung einer Anweisung als Befehl, während es bei dieser Anweisung nicht gestattet ist.

Illegal function call (5)
Es wird mit einem falschen Argument eine BASIC- Funktion aufgerufen. Diese Meldung tritt auf bei:

1 Array-Index(e) negativ oder außerhalb des Array- Bereichs;
2 negativem Argument bei LOG oder SQR;
3 negativer Mantisse mit realem Exponenten;
4 Gebrauch der USR-Funktion ohne spezifizierte Startadresse;
5 verkehrtem Argument bei MID$, LEFT$, RIGHT$, INP, OUT, WAIT, PEEK, POKE, TAB, SPC, STRING$, SPACE$, INSTR, ON...GOTO.
6 Ausführung eines grafischen Befehls, während der Bildschirm sich nicht im grafischen Modus befindet.

Input past end (55)
Man versucht, Daten einzulesen, während alle Daten bereits eingelesen sind. Kann auch auftreten, wenn die Datei leer ist, d.h. überhaupt keine Daten enthält.

Internal error (51)
Der BASIC-Interpreter selbst enthält einen Fehler, z.B. durch Beschädigung des ROM-Speichers.

Line buffer overflow (25)
Der Puffer zur Eingabe einer Zeile ist voll.

Missing operand (24)
Irgendeine der Angaben in einem Ausdruck ist nicht in Ordnung.

NEXT without FOR (1)
Es wurde eine NEXT-Anweisung ohne zugehörige FOR-Anweisung gefunden.

NO RESUME (21)
Die angewandte Fehlerberichtigungsroutine enthält keine RESUME-Anweisung.

OUT of DATA (4)
Man versucht, mit einer READ-Anweisung Daten zu lesen, die nicht in der DATA-Liste enthalten sind.

Out of memory (7)
Für den bereitstehenden Speicherraum ist das Programm zu groß, oder das Programm enthält zu viele Variablen, GOSUB's oder 'eingegliederte' FOR- Schleifen.

Out of string space (14)
Verfügbarer Speicherraum für Strings ist voll. Mit der CLEAR-Anweisung kann der Speicherraum erweitert werden.

Overflow (6)
Eine Berechnung ergibt eine zu große Zahl (sog. Overflow). Es ist auch möglich, daß ein angegebener Wert außerhalb des erlaubten Bereichs fällt.

Redimensioned array (10)
Man versucht, mit DIM zweimal die gleiche Variable zu spezifizieren. Es kann auch vorkommen, wenn auf eine Anweisung, in der bereits eine Array-Variable vorkommt eine DIM-Anweisung folgt.

Rename across disk (71)
Mit dem NAME-Befehl wird ein neuer Datei-Name zugewiesen, bei dem die Antriebsbezeichnung nicht übereinstimmt mit dem Antrieb, in dem sich die Diskette mit der entsprechenden Datei befindet.

RETURN without GOSUB (3)
Es wird eine RETURN-Anweisung gefunden ohne zugehörigen Sprung zu einer Subroutine durch eine GOSUB-Anweisung.

RESUME without error (22)
Es wird eine RESUME-Anweisung ohne zugehörige Fehlerberichtungsroutine gefunden.

String formula too complex (16)
Der String-Ausdruck ist zu kompliziert und muß daher in kleinere Stücke verteilt werden.

String too long (15)
String länger als 255 Zeichen.

Subscript out of range (9)
Verwendung von Array-Indexnummern, die außerhalb des spezifizierten Bereichs liegen.

Syntax error (2)
Allgemeine Fehlerbezeichnung: der Ausdruck entspricht nicht den BASIC-Regeln.

Too many files (67)
Man versucht eine neue Datei zu schaffen, während bereits 255 Dateien vorhanden sind.

Type mismatch (13)
Man versucht, einer String-Variablen einen numerischen Wert zuzuweisen oder umgekehrt.

Undefined line number (8)
Hinweis auf eine nicht vorhandene Zeilennummer.

Undefined user function (18)
Eine FN-Funktion wird angesprochen, die nicht zuvor mit einer DEF FN-Anweisung definiert worden ist.

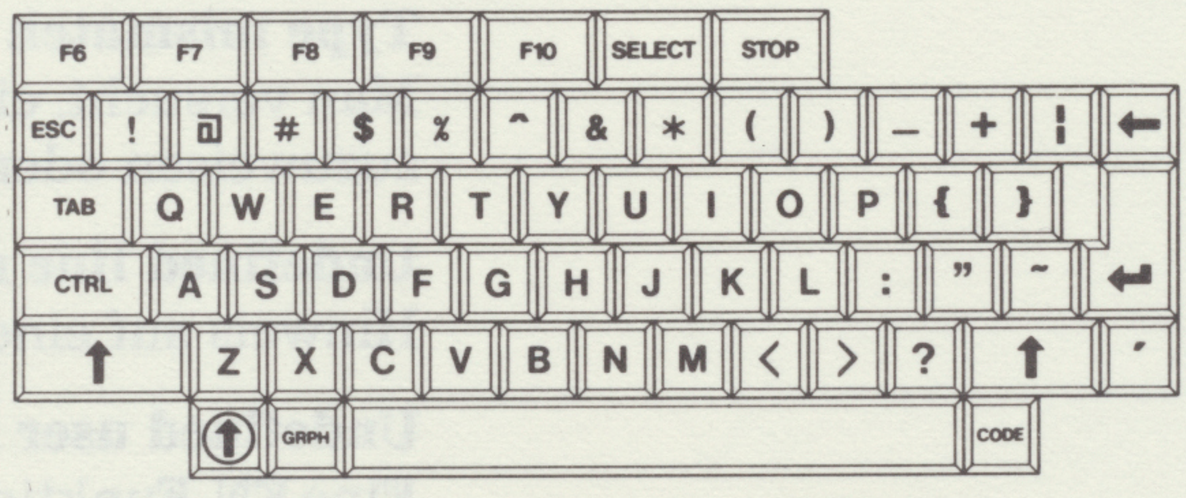
Unprintable error (23, 26-49, 60-255)
Für den angegebenen Code gibt es keine Standard- Fehlermeldung.

Verify error (20)
Feststellung eines Fehlers zwischen dem aufgenommenen und dem im Speicher vorhandenen Programm.

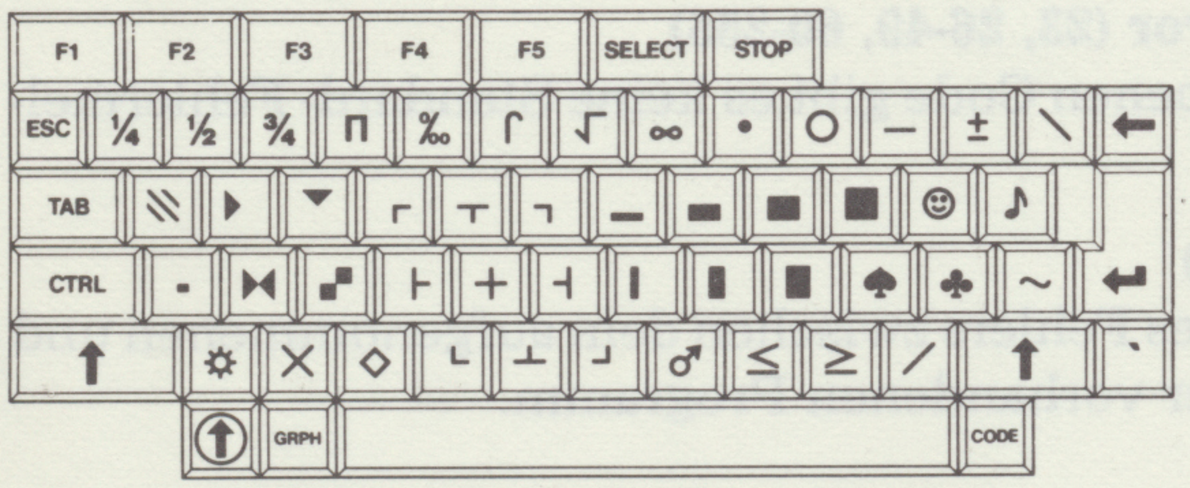
Anhang H Tastatur

Folgende Abbildungen zeigen, welche Symbole man erhält, wenn man die Tasten GRAPH, CODE, SHIFT oder Kombinationen hiervon betätigt. Wir machen den Leser darauf aufmerksam, daß die hier gezeigten Tastaturen sich von der von Ihnen benutzten Tastatur unterscheiden können. In verschiedenen Ländern ist nämlich eine andere Einteilung des Tastenfeldes üblich.
Siehe Handbuch Ihres MSX-Computers.

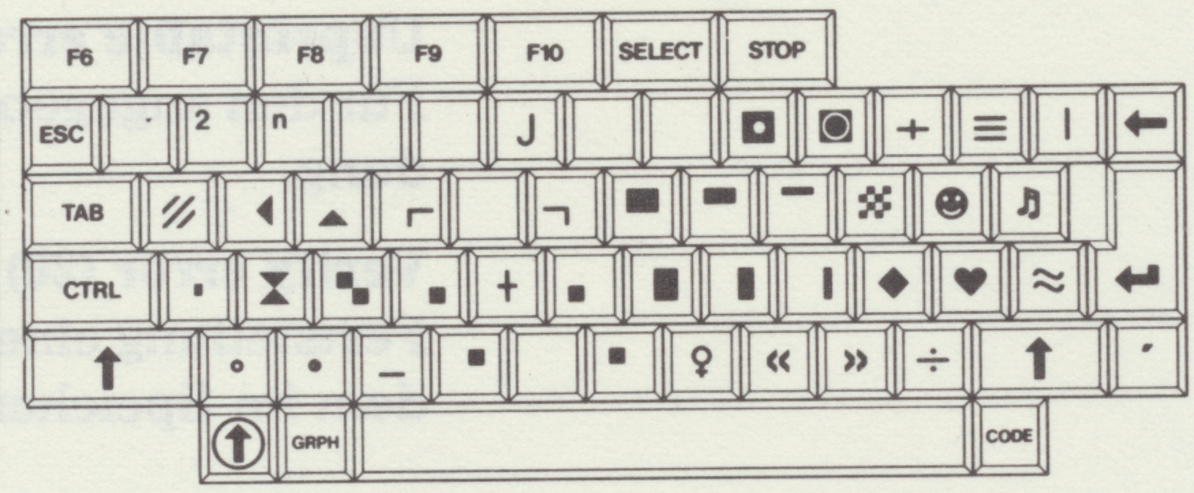
Symbole, die ohne besondere Tastenbetätigung hervorgerufen werden



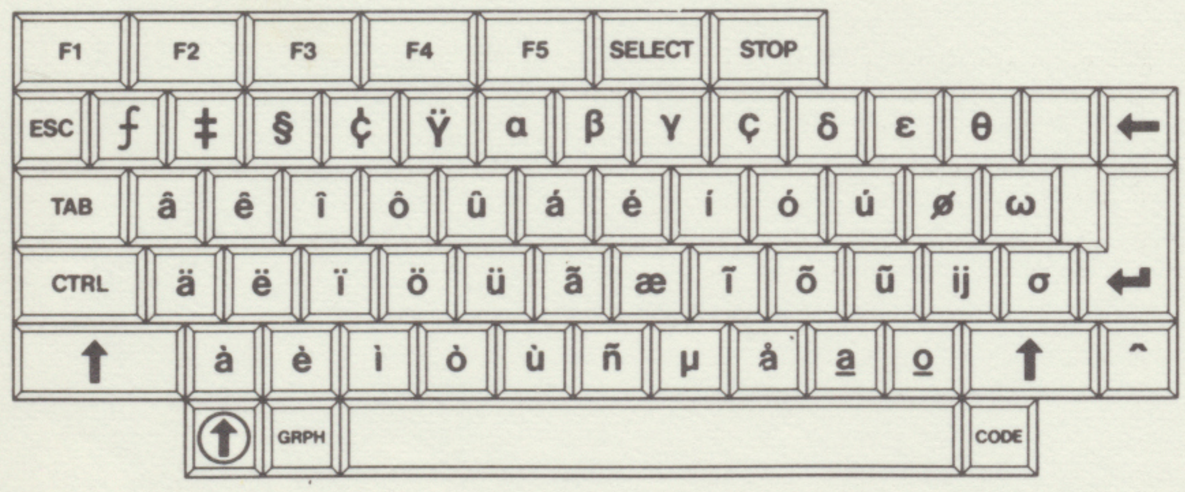
Symbole bei Betätigung der Taste SHIFT



Symbole bei Betätigung der Taste GRAPH



Symbole bei gleichzeitiger Betätigung von GRAPH und SHIFT



Symbole bei Betätigung der Taste CODE

Symbole bei gleichzeitiger Betätigung der Tasten CODE und SHIFT

Anhang I Steuerkodes und Farbkodes

Steuerkodes Eine Reihe von Tasten haben in Verbindung mit der CTRL-Taste eine Sonderfunktion. Drückt man beispielsweise bei eingedrückter CTRL-Taste zugleich auf L, so wird der Bildschirm gelöscht. Diese Handlungsweise schreiben wir als CTRL/L. So besagt etwa auch CTRL/R: Auf R drücken, während CTRL eingedrückt gehalten wird. Nachstehende Übersicht zeigt alle Möglichkeiten:

CTRL/A	Die danach folgende Taste ruft ein Symbol auf aus der alternativen Reihe von Symbolen (siehe Anhang M).
CTRL/B	Versetzt den Cursor zum vorigen Wort.
CTRL/C	Stopp AUTO-Befehl.
CTRL/E	Alle Symbole ab Cursor löschen.
CTRL/F	Versetzt Cursor zum nächsten Wort.
CTRL/G	BEEP.
CTRL/H	Übereinstimmend mit BS-Taste.
CTRL/I	Übereinstimmend mit TAB: zum folgenden Stopp.
CTRL/J	Versetzt Cursor zum nächsten Zeilenanfang.
CTRL/K	Versetzt Cursor in die Ecke oben links.
CTRL/L	Übereinstimmend mit CLS.
CTRL/M	Übereinstimmend mit RETURN-Taste.
CTRL/N	Versetzt Cursor zum Zeilenende (d.h. zur ersten Stelle hinter dem letzten Symbol dieser Zeile).
CTRL/R	Übereinstimmend mit INS-Taste.
CTRL/U	Zeile löschen.
CTRL/\	Wie Cursor-Taste Pfeil nach rechts.
CTRL/]	Wie Cursortaste Pfeil nach links.
CTRL/^	Wie Cursortaste Pfeil nach oben.
CTRL/–	Wie Cursortaste Pfeil nach unten.

Farbkodes Folgende Kodes werden zur Farbkennzeichnung verwendet:

Kode/Farbe	*Kode/Farbe*
0 transparent	8 rot
1 schwarz	9 hellrot
2 grün	10 dunkelgelb
3 hellgrün	11 hellgelb
4 dunkelblau	12 dunkelgrün
5 hellblau	13 magenta
6 dunkelrot	14 grau
7 himmelblau	15 weiß

Anhang J MSX-Disk BASIC

In diesem Anhang wird der Standard MSX-Disk BASIC behandelt.

Allgemeines

MSX-Disk BASIC bietet die Möglichkeit zum Arbeiten mit Disketten. Es wird jedem, der viel mit dem Kassettenrecorder gearbeitet hat, auffallen, wieviel leichter das Arbeiten mit Disketten ist.
Ehe wir auf den Diskettengebrauch tiefer eingehen, müssen wir zuerst einige Begriffe verdeutlichen.
Den ersten Begriff, den wir beschreiben wollen, ist die Datei, auch File genannt.

Datei

Eine Datei (englisch 'file') ist die allgemeine Bezeichnung einer zusammengefaßten Datenmenge, die wir unter einem bestimmten Namen auf Diskette speichern können. Solch eine Datei kann sich beispielsweise auf eine Reihe von Adressen beziehen (Adressendatei), aber auch auf ein BASIC- oder Maschinenkode-Programm.
Wie auch immer der Inhalt beschaffen sein mag, die Datei bekommt auf jeden Fall einen einmaligen Namen, mit dem wir diese Datei dann in Anweisungen oder Befehlen aufrufen können.
Der eigentliche Name einer Datei darf aus höchstens acht Zeichen bestehen. Solche Namensbeispiele sind etwa:

`TEST, TEST1 und EXP1`

Außer Buchstaben und Ziffern dürfen wir auch noch die Zeichen

$ & # % ´ ( ) - @ \ ∧ { } ~ und !

gebrauchen.Neben den eigentlichen Namen können wir noch eine Zusatzangabe machen, um zu verdeutlichen, auf was für

eine Art von Daten (BASIC-Programm, allgemeine Daten usw.) sich die Datei bezieht.
Beispiel:

```
TEST.BAS
```

Hierdurch wird gekennzeichnet, daß es sich um ein BASIC-Programm mit dem Namen TEST handelt.
Wir verwenden im allgemeinen folgende Kennzeichnungen:

Kennzeichnung	*Bedeutung*
ASC	Datei aus ASCII-Codes
ASM	Assembler-Programm
BAK	sogenannte 'back up file' (Kopie einer Datei
BAS	BASIC-Programm
COM	sogenanntes 'utility'-Programm
TEX	Textbestand
PIC	sogenannte 'picture'-Datei

(Anmerkung: Folgende Kennzeichnungen sind unerlaubt: AUX, CON, LST, PRN und NUL.)

Bei bestimmten Befehlen wollen wir mitunter nicht nur ein, sondern eine Sammlung von Programmen angeben. Dafür gebrauchen wir dann Sonderzeichen, die sogenannten 'wild cards'.
So bedeutet:

```
TEST.*
```

alle Programme mit dem eigentlichen Namen TEST und einer beliebigen Kennzeichnung. Und wenn wir angeben:

```
*.BAS
```

so bedeutet dies: 'Alle Programme mit der Kennzeichnung BAS.
Ein weiteres Beispiel:

```
TEST*.BAS
```

bedeutet: 'Alle Programme mit der Kennzeichnung BAS, wobei die ersten 4 Zeichen des eigentlichen Namens bestehen aus TEST'.

Neben dem Zeichen *, das wir als 'wild card' gebrauchen dürfen, können wir auch ? als 'wild card' anwenden. So bezieht sich die Dateikennzeichnung

```
TEST?
```

auf alle Dateien, deren erste 4 Buchstaben übereinstimmen mit TEST, während der fünfte Buchstabe beliebig sein darf. Verfügen wir beispielsweise über die Dateien TEST1, TEST2, TEST3, so werden mit TEST? all diese Dateien angedeutet.

Haben wir nun alles über die Benennungen gesagt? Nein, denn schließlich haben wir noch nicht angegeben, auf welcher Diskette die betreffende Datei gespeichert ist. Wir werden in der Folge hierauf näher eingehen.

Etwas über Drive-Kennzeichnungen und die Befehle COPY, FILES und KILL

Die Diskettenlaufwerke bezeichnen wir im allgemeinen mit dem Namen 'Drive' oder 'Station'. Da wir auch mehr als eine Station an unseren Computer anschließen können, stellt sich dann natürlich die Frage: 'Um welche Station handelt es sich eigentlich?' Die Antwort ist ganz einfach: Wir kennzeichnen die Stationen symbolisch mit den Buchstaben A und B, wenn es sich um zwei Stationen handelt. Mit diesen Buchstaben geben wir dann an, in welcher Station sich die Diskette befindet.
So bedeutet

```
A:TEST.BAS
```

beispielsweise: 'Das Programm TEST.BAS in der Diskette des Laufwerks A'.
Wir geben nun ein Beispiel eines Befehls, in dem die Dateien wie auch die Stationen und auch der COPY-Befehl, mit dem Kopien von Dateien gemacht werden, enthalten sind.
Der Befehl:

```
COPY "A:TEST.BAS" TO "B:"
```

bedeutet: 'Programm TEST.BAS, das sich im Laufwerk A befindet, auf die Diskette in Laufwerk B kopieren.'
Ein weiteres Beispiel:

```
COPY "A:TEST.BAS" TO "B:EXP1.BAS"
```

bedeutet: 'Programm TEST.BAS kopieren und auf Diskette in Laufwerk B unter dem Namen EXP1.BAS speichern.'

Ein weiteres Beispiel, das sich auf den Befehl bezieht, der Dateiübersichten einer Diskette vermittelt. Dieser Befehl trägt den Namen FILES.
So bedeutet:

```
FILES "A:"
```

'Gib eine Übersicht aller Dateien von der Diskette in Station A.'
Wir können auch einen einzigen Namen zum FILES-Befehl eingeben, um auf diese Weise zu ermitteln, ob sich diese Datei auf der Diskette befindet, z.B.:

```
FILES "A:TEST.BAS"
```

Wenn diese Datei tatsächlich vorhanden ist, wird der Name TEST.BAS erneut auf dem Bildschirm wiedergegeben. Ist dies nicht der Fall, so meldet unser Computer:

```
File not found
```

was bedeutet, daß die betreffende Datei nicht vorgefunden wurde.

Noch ein Beispiel, wie wir Dateien löschen können. Hierfür gebrauchen wir den Befehl KILL.
So bedeutet

```
KILL "A:TEST.BAS"
```

'Programm TEST.BAS auf der Diskette im Laufwerk A löschen.'
Und was kann nun folgender Befehl bedeuten:

```
KILL "A:*.*"
```

Genau: 'Alle Programme löschen, die sich auf der Diskette in Laufwerk A befinden.'

Wir werden in der Folge noch sehen, wie wir A und B gebrauchen können, wenn wir nur über ein Laufwerk verfügen.

Starten des Systems und noch mehr über die Befehle FORMAT, SAVE und LOAD

Nach dem Anschließen der Diskstation (beim Kauf wird dazu Anschlußinformation erteilt) wird die Spannung eingeschaltet. Auf dem Bildschirm sehen wir dann:

```
Enter date (M-D-Y):
```

Dies bedeutet, daß wir jetzt ein Datum eingeben sollen, beispielsweise für den 12. Februar 1986:

```
02-12-86
```

Je nach Ausführung Ihres MSX-Computers kann das Datum in einer anderen Reihenfolge erfragt werden, z.B.: (D-M-Y).
Anschließend erfolgt die Mitteilung, daß MSX Disk BASIC jetzt aktiv ist.
Wir nehmen nun noch eine ungebrauchte Diskette, aber die legen wir noch nicht ein. Solch eine ungebrauchte Diskette müssen wir zuerst noch vorbehandeln, und das bezeichnen wir als Formatieren. Erst nach solch einer Vorbehandlung können wir anschließend Dateien auf der Diskette speichern.
Wir geben zu diesem Zweck ein:

```
_FORMAT
```

oder aber

```
CALL FORMAT
```

und nach Betätigung der Return-Taste erscheint folgende Mitteilung:

```
Drive name? (A,B)
```

Anschließend müssen wir angeben, in welches Laufwerk wir

die zu formatierende Diskette einlegen wollen. Wir tippen beispielsweise A, wenn wir die Diskette in Laufwerk A einlegen wollen, also: A (drücken aber noch nicht auf die Return-Taste).
Nun erst legen wir die zu formatierende Diskette in das angegebene Laufwerk. Auf dem Bildschirm lesen wir jetzt:

```
Strike a key when ready
```

Mit dieser Mitteilung wird uns gesagt, daß wir nach dem Einlegen der Diskette in das Laufwerk eine beliebige Taste betätigen müssen; dies ist das Zeichen für den Computer, daß er mit dem Formatieren beginnen kann. Wir hören nun ein sich wiederholendes Geräusch. Sobald die Diskette formatiert ist, erscheint auf dem Bildschirm:

```
Format complete
Ok
```

Zur Kontrolle dessen, ob wir nun tatsächlich eine Diskette haben, auf der wir beispielsweise Programme abspeichern können, geben wir ein kurzes Versuchsprogramm ein:

```
10 PRINT "MSX DISK BASIC"
```

Zur Speicherung dieses Versuchsprogrammes verwenden wir den SAVE-Befehl und tippen:

```
SAVE "A:TEST.BAS"
```

womit gesagt wird: 'Speichere das Programm unter dem Namen TEST.BAS auf die Diskette im Laufwerk A.'
Nach Betätigung der Return-Taste schnurrt unser Computer kurz. Ist das Programm nun wirklich auf der Diskette gespeichert?
Dies probieren wir aus, indem wir das vorhandene Programm, das sich noch im Speicher befindet, mit dem NEW-Befehl löschen. Nun laden wir das Programm von der Diskette mit dem LOAD-Befehl:

```
LOAD "A:TEST.BAS"
```

Nach dem Drücken der Return-Taste schnurrt die Station erneut. Geben wir anschließend den LIST-Befehl, so wird erkennbar, daß sich unser Programm tatsächlich im Speicher des Computers befindet.
Zur Ermittlung, ob sich die Datei TEST.BAS tatsächlich auf der Diskette in Station A befindet, hätten wir auch natürlich den bereits erwähnten FILES-Befehl geben können (tun Sie das!):

```
FILES A:*.*
```

oder noch kürzer

```
FILES
```

Wir verfügen über nur ein Laufwerk

In den meisten Fällen dürfte wohl nur ein Laufwerk zur Verfügung stehen. Was tun wir dann?
In diesem Fall haben die Laufwerknamen A und B keinerlei Bezug auf die Laufwerke, sondern auf zwei Disketten. Dies können wir mit folgendem Beispiel illustrieren:
Wenn wir den CALL FORMAT-Befehl geben, fragt der Computer nach dem Laufwerknamen (A oder B). Wählen wir nun den Namen B, so erhält die Diskette, die sich im Laufwerk befindet, die Kennzeichnung B. Dies bedeutet u.a., daß wir hier keine Programme einspeichern können, wenn wir mit dem SAVE-Befehl nun Laufwerk A angeben.
Ein weiteres Beispiel hinsichtlich des COPY-Befehls. Gesetzt den Fall, daß wir zwei Disketten (A und B) haben und das Programm TEST.BAS von Diskette A auf Diskette B kopieren wollen und sich zugleich Diskette A noch im Laufwerk befindet. Wir geben nun den folgenden Befehl:

```
COPY "A:TEST.BAS" TO "B:"
```

Nun wird dieses Programm zuerst in den Speicher des Computers geladen, und anschließend gibt der Computer an:

```
Insert diskette for drive B
and strike a key when ready
```

In diesem Fall müssen wir nun Diskette B in das Laufwerk einlegen. Kann das Programm nicht auf einmal kopiert werden, so muß es schrittweise geschehen. Der Computer gibt jeweils an, ob wir Diskette A oder Diskette B in das Laufwerk einlegen müssen.
Noch ein praktischer Tipp: Auf der Diskette den Namen notieren, also A oder B, denn das beugt vielen Problemen vor.

MSX Disk BASIC bietet zugleich die Möglichkeit zur Arbeit mit sequenziellen Dateien.

Sequenzielle Dateien

Schon in Anhang A haben wir das Prinzip einer sequenziellen Datei dargelegt, so daß wir darauf hier nicht weiter eingehen müssen.
Es gibt jedoch noch einen Punkt - nämlich die APPEND- Funktion - die genauer zu besprechen ist. Mit dieser Funktion können wir eine sequenzielle Datei erweitern, und dies ist bei einer Kassette nicht möglich.
Wollen wir in einer vorhandene Datei weitere Daten einspeichern bzw. die Datei erweitern, so geben wir folgende OPEN-Anweisung:

```
OPEN "Name:DATA" FOR APPEND AS # Nummer
```

Wir wollen einmal davon ausgehen, daß wir mit folgendem klei-

nen Programm eine Datei gemacht haben:

```
10 OPEN "A:DAT" FOR OUTPUT AS #1
20 INPUT X$
30 IF X$="END" THEN CLOSE #1:END
40 PRINT #1, X$
50 GOTO 20
```

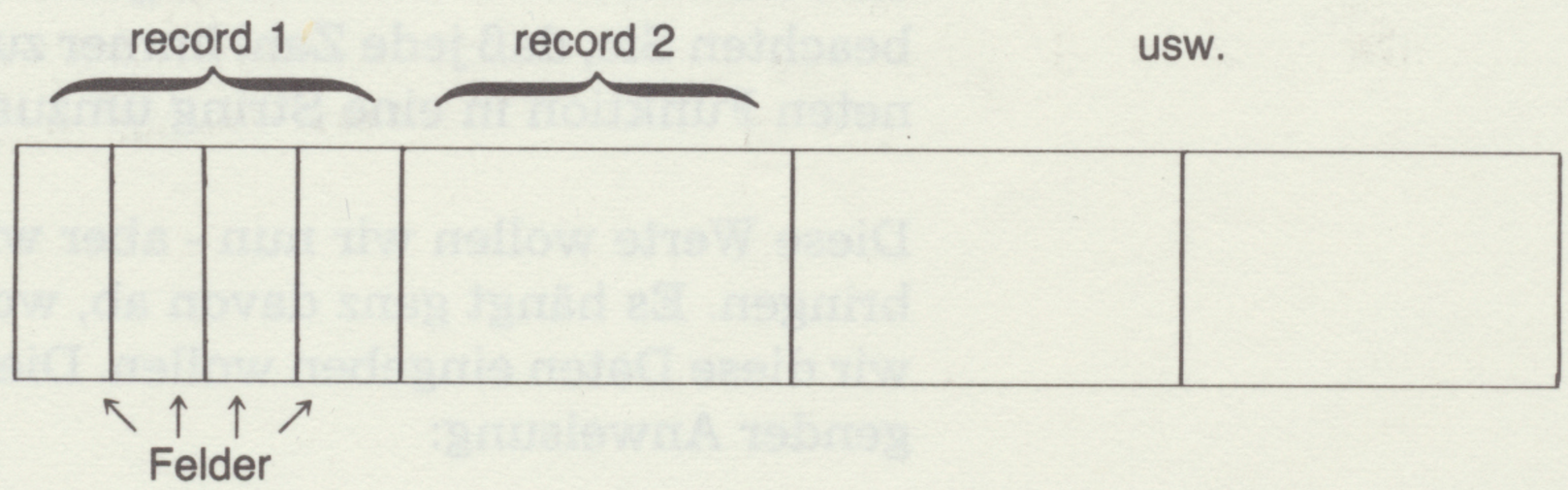
Wenn wir auf diese Weise einige Strings in A:DAT gespeichert haben, so ist auf der Diskette die Datei A:DAT vorhanden. Das folgende Kurzprogramm zeigt, wie wir dieser bereits vorhandenen Datei Daten hinzufügen können:

```
10 OPEN "A:DAT" FOR APPEND AS #1
20 INPUT X$
30 IF X$="END" THEN CLOSE #1:END
40 PRINT #1, X$
50 GOTO 20
```

Der Unterschied zum vorigen Programm ist praktisch nur das Wort APPEND.

Random access file

Ein sogen. 'random access file' ist eine Datei mit der besonderen Eigenschaft, daß wir innerhalb bestimmter Positionen der Datei Daten einlesen oder auslesen können. Um dies zu begreifen, wollen wir folgende Darstellung betrachten:



In dieser Darstellung sehen wir einen Zusammenschluß sogenannter 'Records', die jeweils wieder unterteilt sind in sogenannte 'Felder' (FIELDS).
Dies bedeutet, daß jedes 'Record' eine in Byte ausgedrückte feste Länge hat (LEN) und diese Länge aus Feldern mit ebenfalls einer feststehenden Länge aufgebaut ist. Zur Definierung einer derartigen Random File müssen wir erst LEN angeben, z.B.:

```
OPEN "A:DATA" AS #1 LEN=34
```

In diesem Beispiel hat jedes Record eine Länge von 34 Byte. Da wir bei dieser OPEN-Anweisung nicht den Ausdruck FOR INPUT, OUTPUT oder APPEND mitgeben, weiß der Computer automatisch, daß es sich um eine Random-Datei handelt.

Wir müssen außerdem angeben, wie jedes Feld aufgebaut ist und zugleich, mit welchem symbolischen Namen wir auf solch ein Feld hinweisen.
Beispiel:

```
FIELD #1, 20 AS A$, 8 AS D$, 4 AS S$, 2 AS I$
```

Wir sehen, daß bei unserem Beispiel offenbar jedes Record aus 4 Feldern mit den Namen A$, D$, S$ und I$ besteht, die wiederum die angegebene Menge Bytes enthalten. Die Gesamtmenge der Bytes muß natürlich übereinstimmen mit der Anzahl, die wir bei LEN angegeben haben (Richtigkeit kontrollieren!).

Damit ist gewissermaßen das Gerüst der Datei festgelegt, und wir können sehen, wie wir die Daten in solch einer Datei unterbringen. Hierfür gebrauchen wir die Anweisungen LSET und RSET. LSET dient immer zur Speicherung von Strings und RSET zur Speicherung von Zahlen.
Nachstehendes Beispiel zeigt uns die verschiedenen Möglichkeiten:

```
LSET A$ = WA$
RSET D$ = MKD$ (WD#)
RSET S$ = MKS$ (WS!)
RSET I$ = MKI$ (WI%)
```

WA$ ist hierbei eine String, WD# eine doppelt genaue Zahl, WS! eine Zahl mit einfacher Genauigkeit und WI% ein Integer. Bitte beachten Sie, daß jede Zahl immer zuerst mit einer dazu geeigneten Funktion in eine String umzusetzen ist.

Diese Werte wollen wir nun - aber wie? - in einer Datei unterbringen. Es hängt ganz davon ab, wo, d.h. in welches Record, wir diese Daten eingeben wollen. Dies tun wir dann gemäß folgender Anweisung:

```
PUT #1, Recordnummer
```

z.B.

```
PUT #1,3
```

Bitte beachten, daß hiernach alle Daten, die wir A$, D$, S$ und I$ mittels LSET und RSET zugewiesen haben, in der Datei gespeichert sind, so daß wir A$, D$, S$ und I$ auf gleiche Weise wieder neue Daten zuweisen können, die wieder in ein anderes Record eingespeichert werden können.

Eine Datei müssen wir schließlich bei Beendigung eines Programmes immer abschließen, also:

```
CLOSE #1
```

Beim Auslesen definieren wir die Random-Datei wieder auf gleiche Weise, wie bereits erwähnt.
Wenn wir in einem Programm einer Datei Schreib- sowie Leseaktionen haben, brauchen wir die Datei nur einmal auf die bekannte Weise zu definieren.
Das Auslesen geschieht mit der GET-Anweisung.
So wird mit

```
GET #1,3
```

der Inhalt der Felder von Record 3 jetzt A$, D$, S$ und I$ zugewiesen. Diese Werte können wir beispielsweise drucken mit

```
PRINT A$
PRINT CVB (D$)
PRINT CVS (S$)
PRINT CVI (I$)
```

Mit den Anweisungen CVD, CVS und CVI kann man dann wieder 'saubere' Zahlen bekommen.

Übersicht der Befehle

Nachstehend sind alle Befehle/Anweisungen zu finden.

BLOAD "<Dateispezifizierung>"[,R][,Verschiebung]
BLOAD "<Dateispezifizierung>"[,S][,Verschiebung]
Zum Laden eines Programmes in Maschinensprache in den Speicher. Mit [,R] wird das Programm automatisch gestartet. Mit [,S] wird der Video-RAM-Speicher gekennzeichnet.
Beispiele: `BLOAD "VBD"` und `BLOAD "B:V2"`

BSAVE "<Dateispezifizierung>",<Beginnadresse>,<Endadresse>[,Beginnadresse zum Starten des Programmes]
BSAVE "<Dateispezifizierung>",<Beginnadresse>,<Endadresse>[,Beginnadresse zum Starten des Programmes][,S]
Zum speichern eines Programmes in Maschinensprache. Mit [,s] wirdt der Video-RAM-Speicher gekennzeichnet.
Beispiele: `BSAVE "TIMER", &HC000,&HCFFF`

CLOSE [#Dateinummer>[,[#<Dateinummer...>]]
Zum Schließen von Dateien.

COPY "<Dateikennzeichnung>"TO"<Dateikennzeichnung>"
Zum Kopieren von Dateien. Beispiele:

```
COPY "A:TEST.ASC" TO "B:"
COPY "A:VBD" TO "B:"
```

Bei einem System mit nur einer Diskettenstation geht man davon aus, daß die benutzten Disketten die Namen A und B haben. Man muß diese Kassetten dann in dieser einen Station austauschen.
Die Mitteilung

```
Insert diskette for drive B:
and strike a key when ready
```

bedeutet:
Diskette B einlegen
und danach eine beliebige Taste drücken.

DSKO<Nummer der Station>,<Nummer des logischen Sektors>
Speicherinhalt (gekennzeichnet durch den Inhalt der Speicheradressen &HOF351 und &HOF352) in den spezifizierten Sektor schreiben.

FIELD [#]<Dateinummer>,<Feldbreite>AS <String-Variable>
Hiermit bekommen wir Zugang zum Pufferspeicher einer Random-Datei. Wir können damit in einem BASIC- Programm den Pufferspeicher lesen oder beschreiben, indem wir die String-Variablen anwenden, die im AS-Teil genannt sind.
Ein Beispiel:

```
FIELD #1,20 AS A$, 10 AS B$
```

Hiermit werden die ersten 20 Positionen des Puffers von Datei 1 zugewiesen an A$ und die folgenden 10 Positionen an B$. Führen wir nun einen GET-Befehl aus, so können wir in A$ und B$ das gelesene Record aufrufen. Wir können ein neues Record schreiben, indem wir mit einem LSET oder RSET-Befehl A$ und B$ einen neuen Wert geben und anschließend einen PUT-Befehl geben; siehe Einleitung. FIELD selbst liest nicht von oder schreibt nicht auf die Diskette.
Die Summe der Feldbreiten in einem FIELD-Befehl darf nicht größer sein, als die Recordlänge der Datei (mit OPEN-Befehl mitgeteilt).

FILES ["<Dateispezifizierung>"]
LFILES ["<Dateispezifizierung>"]
Inhaltsverzeichnis aller Dateien auf einer Diskette. Mit LFILES wird dieses Inhaltsverzeichnis mit dem Drucker ausgeschrieben. Beispiele:

`FILES` : Übersicht aller Dateien
`FILES"A:*.BAS"`: Übersicht aller BASIC-Dateien auf A:

CALL FORMAT oder _FORMAT
Zur Formatierung einer bisher ungebrauchten Diskette. Nach dem Befehl CALL FORMAT erfolgt die Frage, in welchem Laufwerk sich die betreffende Diskette befindet.
Vorsicht: Der FORMAT-Befehl löscht den Inhalt einer bereits benutzten Diskette vollständig!

GET [#]<Dateinummer>[,<Recordnummer>]
Hiermit können wir ein Record einer Random-Datei in den Pufferspeicher dieser Datei einlesen. Den Recordinhalt können wir anschließend in einem Programm anwenden, wenn für die Datei durch Anwendung der String-Variablen, die in dem FIELD-Befehl genannt sind, ein FIELD-Befehl gegeben wird.
Jedes Record in einem Random-Speicher hat eine Nummer. Nach dem Einlesen eines Record wird intern die Nummer des Record festgehalten, und bei einem nächsten GET-Befehl wird das folgende Record automatisch eingelesen. Wenn jedoch in dem GET-Befehl eine Recordnummer genannt ist, wird das Record mit dieser Nummer eingelesen.
Die Nummer des zuletzt eingelesenen Records kann man mit der Funktion LOC - siehe dort - finden.
Beispiel:

```
10 OPEN "VBD.DAT"AS#1
20 FIELD #1,2 AS A$,10 AS B$
30 FOR K%=1 TO 10
40 GET #1,K%
50 PRINT CVI(A$);B$
60 NEXT
70 CLOSE #1
80 END
```

INPUT #<Dateinummer>,<Variablenverzeichnis>
Siehe Anhang D
Anmerkung: INPUT # funktioniert nur bei Dateien, die im FOR INPUT-Modus (siehe (OPEN-Befehl) eröffnet sind. Gleiches gilt für den Befehl LINE INPUT # und die Funktion INPUT$.

KILL "<Dateikennzeichnung>"
Zum Löschen einer Diskettendatei.
Beispiel: `200 KILL "X1.DAT"`

LINE INPUT #<Dateinummer>,<String-Variable>
Siehe Anhang D.

LOAD <Dateikennzeichnung>[,R]
Laden einer Datei von der Diskette in den Speicher. Mit ,R wird das Programm sofort nach dem Einlesen gestartet.
Beispiel: `LOAD "PONG",R`

LSET <String-Variable>=<Stringausdruck>
RSET <String-Variable>=<Stringausdruck>
Mit diesen Befehlen können wir Daten in den Puffer einer Random-Datei schreiben. LSET sorgt dafür, daß die String an der

rechten Seite mit Leerpositionen ergänzt wird, so daß die String links im Feld steht. Mit RSET wird die String ganz nach rechts verschoben. Beispiel:

```
140 FIELD #1, 10 AS A$
150 LSET A$ = MKS$(AMT)
```

Im Puffer von Datei 1 erscheint nun die Funktion MKS$(AMT). Der Puffer kann jetzt als Record mit dem PUT- Befehl (siehe dort) zur Datei geschrieben werden.

MAXFILES =<Anzahl>
Gibt die Höchstanzahl Dateien an, die zugleich geöffnet sein können. z.B. MAXFILES=4.
Es können nur 6 Dateien zugleich in einem Programm geöffnet sein.

MERGE <Dateispezifizierung>
Siehe Anhang D.

NAME <alter Name> AS <neuer Name>
Zur Umbenennung von Dateien.
Beispiel: `NAME "VBD" AS "EXPL"`

OPEN "<Dateispezifizierung>"[FOR<Modus>]AS[#] <Dateinummer>[LEN=<reclen>]
Allgemeine Anweisung, um Diskette I/O möglich zu machen.
<Modus> kann sein: FOR OUTPUT (sequenzielles Output)
FOR INPUT (sequenzielles Input)
FOR APPEND (hinzufügen)
<reclen> gibt Record-Länge bei Random-Dateien an.
Beispiel: `10 OPEN "EXPL" FOR INPUT AS #1`
Bei Nichtangabe des <Modus> wird die Datei als eine Random-Datei definiert.

PRINT #<Dateinummer>,[USING"<Schreibkode>";<Reihe Ausdrücke>]
Siehe Anhang D.
Der PRINT #-Befehl wirkt nur in Dateien, die im FOR OUTPUT- oder FOR APPEND-Modus geöffnet sind (siehe OPEN-Befehl).

PUT [#]<Dateinummer>[,<Recordnummer>]
Hiermit können wir das im Pufferspeicher Vorhandene der genannten Datei zur Datei schreiben. Die Nummer, unter der das Record weggeschrieben wird, ist entweder 1 größer als die Nummer des zuletzt gelesenen oder geschriebenen Records (wenn keine Recordnummer angegeben worden ist) oder entspricht der genannten Nummer.

Siehe auch FIELD, GET, LSET, RSET und OPEN.
Beispiel:

```
10 OPEN "EXPL.DAT"AS#1
20 FIELD #1,2 AS A$,10 AS B$
30 FOR K%=1 TO 10
40 INPUT N%,S$
50 LSET A$=MKI$(N%)
60 RSET B$=S$
70 PUT #1,K%
80 NEXT
90 CLOSE #1
100 END
```

RUN [<Zeilennummern>]
RUN <Dateispezifizierung>
Siehe Anhang D

SAVE <Dateispezifizierung>[,A]
Zur Diskettenspeicherung einer Datei. Mit ,A wird die Datei im ASCII-Format gespeichert.
Beispiel: `SAVE "EXPL",A`

CALL SYSTEM oder _SYSTEM
Hiermit kommt das System unter die Kontrolle des MSX-DOS Operating Systems.

Funktionen

In der Folge findet man eine Übersicht der spezifischen MSX Disk BASIC-Funktionen.

CVI (<2-Byte String>)
CVS (<4-Byte String>)
CVD (<8-Byte String>)
Diese Funktionen dekodieren die entsprechenden Strings in integere (CVI), wirkliche (CVS) oder wirkliche Werte mit doppelter Genauigkeit (CVD). Die Strings müssen gemäß den umgekehrten Funktionen (MKI$, MKS$ und MKD$, siehe dort) kodiert sein oder von einer Random-Datei eingelesen werden. Diese Funktionen können wir anwenden, um Zahlen zurückzufinden, die in einer Random-Datei gespeichert sind. In einem FIELD-Befehl können nur String-Variablen angegeben werden; mit diesen Funktionen können wir Zahlen, die in String-Form codiert sind, aus einem Record lesen.

DSKF (<Nummer der Disketteneinheit>)
Diese Funktion gibt die Raummenge an, die auf einer Diskette übrig ist. Die Diskettenstationen sind folgendermaßen numeriert:

0 = fehlende Station
1 = Station A
2 = Station B

DSKI$ (<Nummer der Diskettenstation>, <Sektorennummer>)
Diese Funktion lädt bei Aufruf den entsprechenden Sektor in den Speicherteil, der durch die Speicherplätze &HOF351 und &HOF351 angewiesen wird.

EOF (<Dateinummer>)
Siehe Anhang D
EOF kann nur für Dateien angewandt werden, die mit dem FOR INPUT-Modus geöffnet sind (siehe OPEN-Befehl).

INPUT$ (<n>[,[#]<Dateinummer>])
Siehe Anhang D.
INPUT$ kann nur für Dateien angewandt werden, die im FOR INPUT-Modus geöffnet sind (siehe OPEN-Befehl).

LOC (<Dateinummer>)
Diese Funktion erbringt ein Resultat, das von dem Modus abhängig ist, in dem die Datei geöffnet ist. Handelt es sich um eine Random-Datei, so ergibt die Funktion die Nummer des zuletzt eingelesenen oder weggeschriebenen Records: siehe GET. Bei andersartigen Dateien ergibt die Funktion die Anzahl behandelter (eingelesener oder weggeschriebener) Bytes.

LOF (<Dateinummer>)
Diese Funktion vermittelt die Länge (in Bytes) der entsprechenden Datei.

MKI$ (<Integer-Ausdruck>)
MKS$ (<Real-Ausdruck>)
MKD$ (<Real-Ausdruck mit doppelter Genauigkeit>)
Diese Funktionen vermitteln eine String, die die kodierte Version der genannten Zahl enthält. Die String kann mit den umgekehrten Funktionen CVI, CVS und CVD dekodiert werden; siehe dort.
Die durch Anwendung dieser Funktionen erhaltenen Strings können in ein Record einer Random-Datei geschrieben werden.

VARPTR (#<Dateinummer>)
Neben der normalen VARPTR-Anwendung (siehe Anhang D) können wir diese Funktion auch zum Finden der Adresse des Dateisteuerblocks der betreffenden Datei anwenden. Diese Adresse kann dann auch an eine Maschinensprache-Routine weitergeleitet werden.

Anhang K MSX Quick Disk BASIC

Allgemeines

MSX Quick Disk ist eine sehr elegante Zwischenlösung für diejenigen, die einen Kassettenrecorder nicht länger gebrauchen wollen, aber denen eine 'gewöhnliche' Diskettenstation zu teuer ist.
Wir verwenden 2.8 inch Disketten, die pro Seite 64 K Byte speichern können. Auffälligstes Kennzeichen von MSX Quick Disk BASIC dürfte sicher sein, daß soviel Übereinstimmung in der Anwendung mit MSX Disk BASIC herrscht. Ein wesentlicher Unterschied ist jedoch, daß MSX Quick Disk BASIC keine Möglichkeit bietet, mit Random Access Dateien zu arbeiten - aber Unterschiede gibt es nun einmal.
Wir werden im Folgenden zuerst eine kurze Einführung für Anwender geben, die erstmals mit dem System arbeiten. Anschließend werden wir auf übliche Weise eine Übersicht aller Befehle geben, die Quick Disk BASIC bietet.

Erste Bekanntschaft

Das System besteht aus dem Laufwerk, d.h. dem Teil, in den wir die Diskette einlegen, sowie aus einem breiten Stecker, den wir in den breiten Anschluß unseres Computers stecken (mit dem Text des breiten Steckers nach vorn, d.h. so, daß der Anwender den Text sieht).
Zur Sicherheit: Während dieser Tätigkeit Spannung abschalten, sowohl vom MSX-Computer als auch vom Quick Disk System!
Nach Einschalten der Spannung geben wir eine noch ungebrauchte Diskette in die Station. Man tut dies, indem man (bei geöffnetem Deckel) die Diskette in den Schlitz schiebt und anschließend den Deckel schließt.
Wir müssen nun wieder als erstes den FORMAT-Befehl geben (siehe auch Anhang J). Damit wird die Diskette vorbehandelt, so daß wir anschließend die Daten bzw. Programme auf der Diskette speichern können.

Wir tippen zu diesem Zweck:

```
CALL QDFORMAT
```

oder das gleichwertige

```
– QDFORMAT
```

Nach Betätigung der Return-Taste erscheint der Text:

```
ARE YOU SURE (Y/N)?
```

(übersetzt: Sind Sie ganz sicher (J/N))?
Dies ist eine zusätzliche Warnung, denn wir dürfen nicht vergessen, daß beim Formatieren bzw. der Ausführung des FORMAT-Befehls alle eventuell noch auf der Diskette vorhandenen Programme gelöscht werden.
Wenn jedoch die Diskette wirklich ungebraucht ist, haben wir nichts zu befürchten und drücken daher auf Y.
Anschließend lesen wir auf dem Bildschirm:

```
Now formatting
```

und, wenn alles gut geht, danach:

```
Complete
OK
```

Nachdem nun unsere Diskette formatiert ist, können wir hierauf wirklich Daten speichern.
Wir wollen die häufigsten Handlungen einmal mit nachstehendem Programmbeispiel zeigen.
Unser Programmbeispiel lautet:

```
10 PRINT "QUICK"
20 PRINT "DISK"
```

Wir tippen dieses Programm und wollen es anschließend auf der Diskette speichern. Dazu erteilen wir folgenden Befehl:

```
CALL SAVE ("TEST")
```

Nach Betätigung der Return-Taste schnurrt die Station einen Augenblick und macht so bemerkbar, daß der Befehl ausgeführt wird. Nun Bildschirm löschen mit NEW.
Um unser Programm wieder zu sehen, tippen wir:

```
CALL LOAD ("TEST")
```

Nach dem LIST-Befehl erscheint erneut unser ursprüngliches Programm.
Aus der Übersicht der Befehle ist ersichtlich, daß wir zugleich eine Nummer der Disk-Station nennen können. Wir hätten so bei 'SAVE' beispielsweise auch eingeben können:

```
CALL SAVE ("QDO:TEST")
```

Hierdurch wird angegeben, daß es sich um das Laufwerk Nr. 0 handelt.
Wenn wir jedoch nur über ein Laufwerk verfügen, was wohl in der Regel der Fall sein dürfte, kann diese genauere Bezeichnung weggelassen werden.

Noch eine Bemerkung zur Namensgebung. Das Prinzip ist hier wieder das gleiche wie bei MSX Disk BASIC. Hinter den eigentlichen Namen (höchstens 8 Zeichen) darf man noch eine Erklärung in 3 Buchstaben setzen, z.B.

```
     TEST.BAS        (BASIC-Programm)
oder EXP.DAT         (Datenbestand)
```

Um einen Überblick der auf einer Diskette gespeicherten Dateien zu bekommen, gibt man den Befehl:

```
CALL QDFILES
```

und in unserem Beispiel erscheint dann:

```
     NAME      ATR        SIZE
("QD0:TEST"    /02        001E
```

Links sehen wir den Namen unseres Programms, zu dem nun automatisch die Laufwerknummer kommt und rechts bei ATR (Attribut) einen Kode, um was für eine Art Datei es sich handelt, zugleich eine Zahl (in Hexadezimal) mit der Größe (in Bytes) der Datei.
Hinsichtlich des Kodes bei ATR gelten folgende Regeln:

Kode	*Bedeutung*
01	in binärem Format gespeichert
02	BASIC-Programm
03	im ASCII-Format gespeichert
0B	Abbildung des Bildschirms

Pro Diskette können wir maximal 20 verschiedene Dateien speichern - sofern wir die Speicherkapazität von 64 KByte nicht überschreiten. Als letztes wollen wir in dieser Einleitung demonstrieren, wie wir eine Datei von der Diskette entfernen können. Dies geschieht mit dem QDKILL-Befehl. Unser Programmbeispiel löschen wir mit

```
CALL QDKILL ("TEST")
```

Eine Warnung ist hier jedoch angemessen: unser Computer kann nur die letzte Datei löschen (die in der Übersicht von QDFILES erscheint). Eine Namensangabe erscheint daher bei QDKILL überflüssig. Hierbei handelt es sich jedoch um eine zusätzliche Sicherheit, um unerwünschtes Löschen zu vermeiden.

Soweit unsere kurze Einleitung. In der Folge vermitteln wir eine kurze Übersicht aller Befehle, die das System bietet. In den Fällen, in denen die Bedeutung mit MSX Disk BASIC übereinstimmt, wird hierauf verwiesen.

Übersicht aller Befehle

CALL SAVE ("[QD[<n>]:]<Dateiname>"[,A]
Mit ,A wird angedeutet, daß die Datei im ASCII- Format gespeichert wird. Dies nimmt im allgemeinen viel mehr Raum ein, als wenn wir ,A weglassen und einen vom System gewählten Speicherkode wählen. Wenn bereits eine Datei gleichen Namens vorhanden ist, erfolgt Fehlermeldung Device I/O Error.

CALL LOAD ("[QD[<n>]:]<Dateiname>"[,R]
Wie LOAD bei MSX-Disk BASIC.
Mit ,R wird angegeben, daß das Programm sofort nach dem Laden gestartet werden muß.

CALL BSAVE ("[QD[<n>]:]<Dateiname>"<Beginnadresse>, <Endadresse>[<Startadresse>]
Wie BSAVE bei MSX-Disk BASIC.
Wenn wir einen Teil von Video-RAM, d.h. einen Teil vom Bildschirm speichern wollen dann, setzen wir hinter die Startadresse noch ,S. Beispiel:

```
CALL BSAVE ("SCRN.DAT",&H0,&H3FFF,S)
```

CALL BLOAD ("[QD[<n>]:]<Dateiname>",[R oder S][,<Verschiebung>]
Wie BLOAD bei MSX-Disk BASIC.

CALL RUN ("[QD[<n>]:]<Dateiname>")
Wie RUN bei MSX-Disk BASIC.

CALL MERGE ("[QD[<n>]:]<Dateiname>")
Wie MERGE bei MSX-Disk BASIC.
Zur Deutlichkeit wollen wir noch anmerken, daß dieser Befehl nur mit einer im ASCII-Code eingespeicherten Datei ausgeführt werden kann.

CALL QDFORMAT [("QD[<n>]:")]
Wie FORMAT bei MSX-Disk BASIC.

CALL QDFILES [("[QD[<n>]:]
Wie FILES bei MSX-Disk BASIC.

CALL QDKILL ("[QD[<n>]:]<Dateiname>")
Wie KILL bei MSX-Disk BASIC, wobei anzumerken ist, daß

nur die letzte Datei gelöscht werden kann (ergibt sich aus der Übersicht bei CALL QDFILES).

CALL CASQD [("[CAS:]<Dateiname>"][,QD[<n>]:<Dateiname>"])]]
Zum Kopieren einer Datei von Kassette auf Quick Disk. Die einfachste Form ist:

```
CALL CASQD
```

Hierbei wird als Name der Name genommen, unter dem die Datei auf Kassette gespeichert wurde.
Wenn man auf diese Weise von Software-Herstellern auf Kassette gelieferte Spiele kopiert, wird man häufig feststellen, daß solch ein Spiel nicht funktioniert. Dies ergibt sich aus dem vom Hersteller vorgenommenen Kopierschutzes.

CALL QDKEY [(<n>)]
Hiermit können wir die Bedeutung der Funktionstasten bestimmen. Beim Einschalten des Computers haben diese Tasten folgende Bedeutung:

F1	_RUN	F6	color 15,4,7
F2	_LOAD	F7	_QDKEY
F3	_BLOAD	F8	_SAVE ("QD0:
F4	list	F9	_BSAVE ("QD0:
F5	run	F10	_QDFILES

Mit dem Befehl CALL QDKEY (0) verändert sich die Bedeutung folgendermaßen:

F1	color	F6	color 15,4,7
F2	auto	F7	_QDKEY
F3	goto	F8	cont
F4	list	F9	list
F5	run	F10	run

Achtung: Auf dem Bildschirm wird häufig eine Abkürzung der Funktionskennzeichnung gegeben!

OPEN "QD[<n>:<Dateiname>"FOR<INPUT oder OUTPUT>AS[#]<Dateinummer>
Wie OPEN bei MSX BASIC. Wir können nur mit sequenziellen Dateien arbeiten. Im Anhang A wird die Arbeitsweise erläutert. Hierzu noch zwei Anmerkungen:

a Es dürfen höchstens 2 Dateien zugleich geöffnet sein.
b Die von MSX-Disk BASIC gebotene APPEND-Möglichkeit wird bei MSX Quick Disk BASIC nicht geboten.

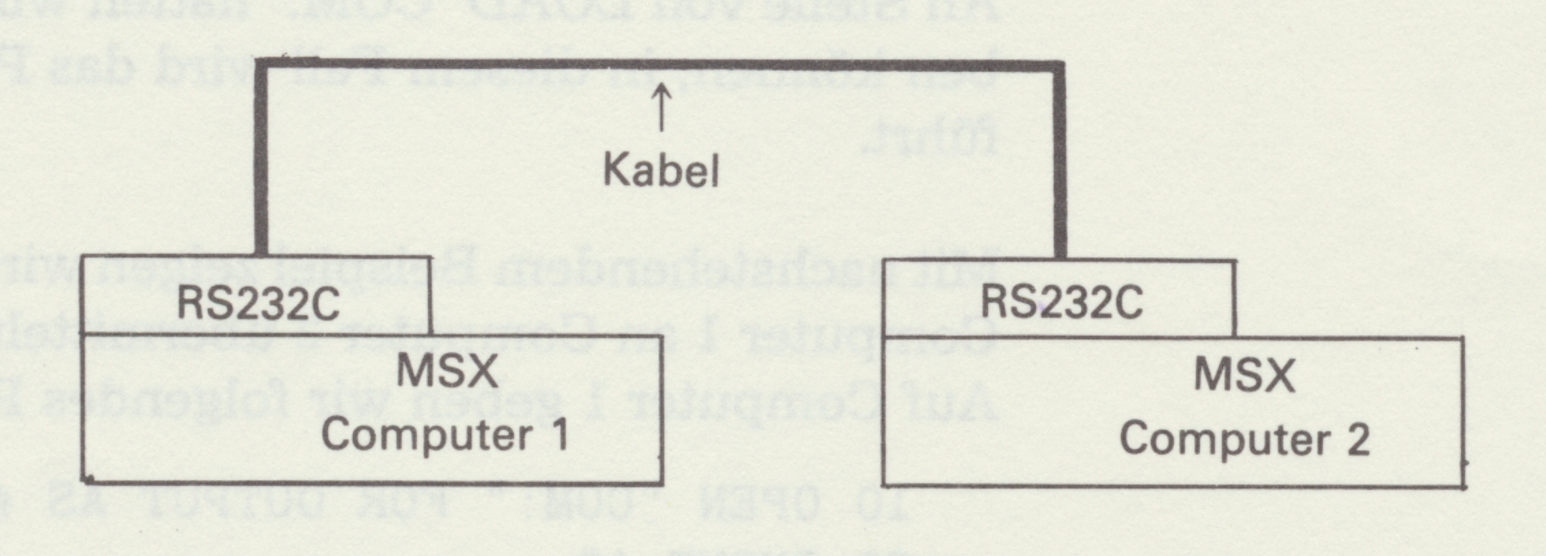
CLOSE [[#]<Dateinummer>[,[#]<Dateinummer>]...]
Wie CLOSE bei MSX-Disk BASIC.

AUTOEXEC
Hierbei handelt es sich um einen besonders reservierten Namen. Wenn wir bei CALL SAVE oder bei CALL BSAVE als Namen des Programmes AUTOEXEC angeben, dann wird beim Einschalten des Systems automatisch dieses Programm gestartet.

Anhang L MSX RS232C-Schnittstelle

Allgemeines

In der Computerwelt spielt die Kommunikation eine wichtige Rolle. Hiermit wird gemeint, daß wir die Information vom einen Ort zum anderen übermitteln können. Man denke beispielsweise an zwei Computer, die miteinander in Verbindung stehen. Es ist doch eine ausgezeichnete Sache, auf einfache Weise Programme oder Datenbestände zu übermitteln. Um dies zu ermöglichen, wurde für den MSX-Computer ein sogenanntes RS232C-Interface, auch Schnittstelle genannt, entwickelt. Dieses Gerät dient dazu, die Information auf ein zur Übertragung geeignetes Format zu bringen.
Bei RS232C handelt es sich um eine sogenannte serielle Schnittstelle, weil die Einsen und Nullen, womit alle Daten festgelegt sind, hier nacheinander, also in Serie (bzw. sequenziell) übermittelt werden.
Natürlich sind wiederum Kontrollsignale nötig, damit dies auf ordentliche Weise abläuft. In der Folge zeigen wir, wie zwei MSX-Computer, die beide mit RS232C-Schnittstellen ausgestattet sind, miteinander kommunizieren können. Schematisch sieht das so aus:



Um es ganz klar zu sagen: in unserem Beispiel kommuniziert ein MSX-Computer mit einem anderen MSX-Computer, aber im allgemeinen könnte es sich bei dem anderen Computer auch um ein beliebiges Gerät handeln, das mit einer RS232C-Schnittstelle ausgestattet ist (man denke beispielsweise an ein Modem,

mit dem man über die Telefonleitung mit einem anderen Computer kommuniziert).
Die Schnittstelle selbst besteht aus einer Cartridge ohne Kabel. An dieser Cartridge befindet sich ein breiter Stecker (ein sogenannter 25-pin Centronics Anschluß).
Im Handel sind Kabel erhältlich, mit denen die beiden Cartridges der zwei Computer (siehe Abb.) miteinander verbunden werden können.

Gehen wir einmal davon aus, daß wir dem Computer 1 das folgende kleine Programm eingeben:

```
10 PRINT "RS232C"
20 PRINT "MSX"
```

Wir erteilen nun folgenden Befehl:

```
SAVE "COM:"
```

Wir wollen dieses kleine Programm über die Kennzeichnung COM festlegen, wobei sich COM auf unsere RS232C- Schnittstelle bezieht. Wenn wir nunmehr auf Computer 2 eingeben:

```
LOAD "COM:"
```

dann wird nach Betätigung der Return-Taste beider Computer das Programm übermittelt. Bitte beachten Sie jedoch, daß wir eine Fehlermeldung (Device I/O error) bekommen, wenn wir nach dem SAVE-Befehl mit dem Eingeben des LOAD-Befehls zu lange warten. Der Grund liegt darin, daß wir mit dem Eingeben des SAVE-Befehls bekanntgeben, daß Computer 1 Daten übermitteln will und sich Computer 1 orientiert, ob Computer 2 empfangsbereit ist (angegeben durch LOAD-Befehl). Dauert dies zu lange, so erscheint die Fehlermeldung.
Dies gilt in der Praxis für jede Kommunikation: Wenn das Empfängergerät nicht seine Empfangsbereitschaft mitteilt, dann wird die Verbindung automatisch unterbrochen.
An Stelle von LOAD "COM:" hätten wir auch RUN "COM:" geben können; in diesem Fall wird das Programm sofort ausgeführt.

Mit nachstehendem Beispiel zeigen wir, wie wir eine Datei von Computer 1 an Computer 2 übermitteln.
Auf Computer 1 geben wir folgendes Programm ein:

```
10 OPEN "COM:" FOR OUTPUT AS #1
20 INPUT A$
30 PRINT #1, A$
40 CLOSE #1
```

Auf Computer 2 wird folgendes Programm eingegeben:

```
10 OPEN "COM:" FOR INPUT AS #1
```

```
20 INPUT #1,A$
30 PRINT A$
40 CLOSE #1
```

Wenn wir nun bei Computer 1 nach dem Starten des Programmes die String MSX eingeben, erscheint auf dem Bildschirm von Computer 2 ebenfalls MSX (wenn wir dem Programm von Computer 2 ebenfalls den Befehl RUN erteilt haben).

Ein weiteres Beispiel:
Wenn wir auf beiden Computern den Befehl

```
CALL COMTERM
```

geben, wird alles, was auf Computer 1 getippt wird, auch auf Computer 2 sichtbar, und umgekehrt wird auf Computer 1 sichtbar, was auf 2 geschrieben wird. In diesem Fall haben wir von beiden Computern ein 'dummes' Terminal gemacht (Schreibmaschine zur Datenübermittlung) und können so sehr einfach Nachrichten austauschen.

Anschluß an ein anderes Gerät als MSX, z.B. einen Computer einer anderen Marke

Allgemeingültig läßt sich das hier nicht behandeln. Dies hängt wiederum damit zusammen, daß allerlei Einstellungen notwendig sind, z.B.:

- Übermittlungstempo
- die sogenannte Paritätskontrolle
- Länge der Bytes
- Anzahl sogenannter Stopp-bits

und oft noch viele andere Gegebenheiten (siehe Übersicht Befehle).
Anfänger müssen wir daher auch warnen: Kommunikation über die RS232C ist nicht einfach, sofern Sie nicht über die notwendigen technischen Kenntnisse verfügen. Nur wenn Sie eine klare Beschreibung haben, wie die Kommunikation durchzuführen ist, kann dies einem Erstanwender gelingen!

1. Weiteres über Anwendung von Datenbeständen

Bei BASIC kann die RS232C-Schnittstelle als Eingabe- und Ausgabemedium verwendet werden. Alle Befehle, die für Datendateien auf Kassette gegeben werden können, sind auch zur Arbeit mit RS232C geeignet. Diese sind:

```
OPEN
PRINT #
PRINT # USING
INPUT #
LINE INPUT #
CLOSE
```

Im OPEN-Befehl muß als Device-Name "COM[<n>]" mitgege-

ben werden. Die Zahl <n> hierin kann meistens weggelassen werden: nötig ist die Angabe der Schnittstellennummer nur, wenn Sie mehr als ein RS232C an Ihren Computer angeschlossen haben.
Einen Dateinamen müssen Sie beim Öffnen einer Datei auf der RS232C-Schnittstelle nicht angeben.
Beispiele:

```
OPEN "COMØ:" AS #1
OPEN "COM:" FOR OUTPUT AS #2
```

Sie sehen, daß beim ersten Beispiel keine Verarbeitungsmethode (FOR INPUT oder FOR OUTPUT) genannt ist. Dies hat zur Folge, daß Sie die auf diese Weise eröffnete Datei zum Einlesen wie zum Wegschreiben gebrauchen können. Auf diese Weise können Sie beispielsweise mit einem anderen MSX-Computer kommunizieren.
Die Anwendung der anderen vorstehend genannten Befehle geschieht, wie an anderer Stelle für Kassettendateien beschrieben wurde.

2. Weiteres über die Anwendung zur Programmspeicherung

Bei der Anwendung der RS232C-Schnittstelle zur Programmspeicherung verfügen Sie über folgende Befehle

```
LOAD "COM[<n>]:"
SAVE "COM[<n>]:"
MERGE "COM[<n>]:"
RUN "COM[<n>]:"
```

Der Device-Name ist in allen Fällen wieder "COM[<n>]:"; hinsichtlich der Zahl <n> gilt das zuvor Besprochene. Ein Dateiname braucht nicht spezifiziert zu werden.
Auf diese Weise übermittelte Programme haben immer ASCII-Format. Dies bedeutet, daß Schlüsselwörter im Programm nicht kodiert werden, sondern Buchstabe für Buchstabe übermittelt werden.

3. Einstellungen der RS232C-Schnittstelle

Bevor zwei Geräte über RS232C miteinander kommunizieren können, müssen die Schnittstellen beider Geräte auf gleiche Weise eingestellt werden. Bei diesen Einstellungen handelt es sich um Baudgeschwindigkeiten, Parität, Bereitschaft ('handshake') und dergleichen. Wenn Sie diese Ausdrücke nicht kennen, hat ein Weiterlesen im Augenblick für Sie keinen Sinn.
Alle Einstellungswerte können durch den Befehl CALL COMINI verändert werden. Die Syntax hiervon ist:
CALL COMINI [([String exp>][,[<Rx Baudgeschwindigkeit>][,[<Tx Baudgeschwindigkeit>][,[<Zeitbegrenzung>]]]])]
Bedeutung der Felder:

<String exp>: siehe unten.

<Rx Baudgeschwindigkeit>: Baudgeschwindigkeit (Bits/Sek.), womit die RS232C-Schnittstelle Zeichen empfängt.
Vorgabe: 1200 Baud

<Tx Baudgeschwindigkeit>: Baudgeschwindigkeit (Bits/Sek.), womit RS232C Zeichen übermittelt.
Vorgabe: 1200 Baud

<Zeitbegrenzung>: Zeit (in Sek.), die der Computer auf Bereitschaft wartet, ehe die Fehlermeldung erfolgt. Zeitbegrenzung=0 bedeutet, daß der Computer unbegrenzt wartet.
Vorgabe: 0

<String exp>: Die Form hiervon ist:
"<knl>:]][<bl>[<pa>[<sl>[<x>[<hs>[<ilf>[<vlf[<ss<]]]]]]]"
Bedeutung:

<kn>: Kanalnummer: Nummer der verwendeten Schnittstelle. Nur nötig, wenn mehr als eine Schnittstelle vorhanden ist.
Vorgabe: 0

<bl>: Byte-Länge: Länge in Bits der auf einmal übermittelten Dateneinheit (Byte). Einzusetzen "5", "7" oder "8".
Vorgabe: "8"

<pa>: Parität
"E" = gerade Parität (Even)
"O" = ungerade Parität (Odd)
"I" = Parität unbeachtet (Ignore)
"N" = keine Parität (No)
Vorgabe: "E"

<sl>: Stopp-Länge: Länge (in Bits) der Pause zwischen der Übermittlung von zwei Bytes.
"1" = 1 Bit
"2" = 1,5 Bits
"3" = 2 Bits
Vorgabe: "1"

<x>:XON/XOFF Steuerung
"X" = steuern
"N" = nicht steuern
Vorgabe: "X"

<hs>: CTS-RTS handshake
"H" = Bereitschaftsmeldung
"N" = keine Bereitschaftsmeldung
Vorgabe: "H"

<ilf>: Einsetzung eines Linefeed-Zeichens, jeweils wenn ein Carriage-Return-Zeichen empfangen wird.
"A" = einsetzen
"N" = nicht einsetzen
Vorgabe: "N"

<slf>: Steuern eines Linefeed-Zeichens nach jedem Carriage-Return.
"A" = kein Linefeed steuern
"N" = Linefeed steuern
Vorgabe: "N"

Übersicht von COMINI

"[0:][8[n[1[X[H[N[N]]]]]]]"

Kanal–Nr. Diese Kennzeichnung darf wegfallen, wenn es nur einen Kanal gibt. Vorgabe = 0.

Datenlänge
"5" - 5 Bits
"6" - 6 Bits
"7" - 7 Bits
"8" - 8 Bits

Paritätsflag
"E" - Even (gerade)
"O" - Odd (ungerade)
"I" - Ignore (unberücksichtigt)
"N" - keine

Stop–bits–(Länge)
"1" - 1 Bit
"2" - 1,5 Bits
"3" - 2 Bits

CTS-RTS–handshake
"H" - Bereitschaftsmeldung
"N" - keine Bereitschaftsmeldung

LF–(Linefeed)–einsetzen
wenn CR empfangen wird
"A" - einsetzen
"B" - nicht einsetzen

LF–(Linefeed)–einsetzen
wenn CR gesendet wird
"A" - einsetzen
"B" - nicht einsetzen

Shift–in/Shift–out–control
(unzulässig wenn)
Datenlänge = 7 Bits
"S" - Control einstellen
"N" - nicht

<ss>: shift in/shift out-Steuerung
"S" = Steuerung
"N" = keine Steuerung
Vorgabe: "N"

Bei einem CALL COMINI-Befehl nicht spezifizierte Einstellungen gehen vom Vorgabewert aus. Einzige Ausnahme dieser Regel ist <Tx Baudgeschwindigkeit>: Ohne Spezifizierung wird ausgegangen vom Wert <Rx Baudgeschwindigkeit>.
Die folgende Tabelle vermittelt nochmals eine Übersicht.

4. Anwendung von RS232C zur Programmsteuerung

Durch Anwendung einiger CALL-Befehle können Sie ein BASIC-Programm automatisch eine Subroutine ausführen lassen, sobald über die RS232C-Schnittstelle ein Zeichen empfangen wird. Das Ergebnis dieser Befehle entspricht voll und ganz dem Ergebnis von ON <Ereignis> GOSUB <Zeilennummer>.
Die Befehle sind:
CALL COMON ("[<n>:]"): funktioniert wie INTERVAL ON
CALL COMOFF ("[<n>:]"): funktioniert wie KEY OFF
CALL COMSTOP ("[<n>:]"): funktioniert wie STRIG STOP
CALL COM ([<n>:]), GOSUB <Zeilennummer>): funktioniert wie ON INTERVAL GOSUB
Feld <n> ist wieder die Nummer der verlangten Schnittstelle und kann weggelassen werden, wenn nur eine Schnittstelle vorhanden ist.

Control-Befehle

CALL COMTERM ["<n>"]
Siehe Einleitung. Mit <n> wird die eventuelle Kanalnummer angegeben.

CALL COMBREAK ["<n>"], <Spezifizierungskode>
Hiermit können sogenannte 'break-Zeichen' übermittelt werden. Der genannte <Spezifizierungskode> wird nun als Zeichen zur Unterbrechung der Verbindung angewandt (Zahl zwischen 3-32767). Dieser Befehl ist nur wirksam, wenn RS232C bereits geöffnet ist.

CALL COMDTR ["<n>"],<0 oder nicht-0>
Hiermit geben wir an, daß unsere Schnittstelle zeitweilig ausgeschaltet ist, jedenfalls wenn der Ausdruck <0 oder nicht-0> gleich 'nicht-0' ist. Dieser Befehl wirkt nur bei geöffnetem RS232C.

CALL COMSTAT ["<n>"],<Name Variable>
Beim Auftreten der Fehlermeldung 'Device I/O error' wird der Variablen ein Wert gegeben, aus dem ersichtlich ist, was even-

tuell falsch ist. Nachstehende Tabelle zeigt die Möglichkeiten. In einer Reihe von Fällen sind die englischen Standardausdrükke übernommen worden.

Bit Nr.	*Bedeutung*
15	Buffer overflow error
	0 - kein buffer overflow
	1 - buffer overflow
14	Time out error (TMENBT)
	0 - nein
	1 - ja
13	Frame error
	0 - nein
	1 - ja
12	Over run error
	0 - nein
	1 - ja
11	Parity error
	0 - nein
	1 - ja
10	Betätigte Taste Control break (BRONBT)
	0 - nein
	1 - ja
9	nicht benutzt
8	nicht benutzt
7	Clear to Send (sendebereit)
	0 - nein
	1 - ja
6	Timer/counter output-2
	0 - timer/counter wird ausgeschaltet
	1 - timer/counter ist eingeschaltet
5	nicht benutzt
4	nicht benutzt
3	Data Set Ready
	0 - nein
	1 - ja
2	break detect
	0 - nein
	1 - ja
1	Ring Indicator
	0 - nein
	1 - ja
0	Carrier Detect
	0 - nein
	1 - ja

Technische Daten Die Kontaktanschlüsse haben folgende Bedeutung (englische Bezeichnungen):

Pin	Bedeutung
1	Frame Ground (F-GND)
2	Transmit Data (TxD)
3	Receive Data (RxD)
4	Request to Send (RTS)
5	Clear to Send (CTS)
6	Data Set Ready (DSR)
7	Signal Ground (S-GND)
8	Carrier Detect (CD)
20	Data Terminal Ready (DTR)
22	Ring Indicator (RI)

Anhang M Aufbau der Symbole

Dieser Anhang vermittelt eine Übersicht aller Symbole (Zeichen) mit den zugehörigen Kodes.
Sie können beispielsweise in der CHR$-Funktion angewandt werden. So bekommt man das Zeichen A durch PRINT "A", sowie auch durch PRINT CHR$(65).
Im Text-Modus (text-mode) 1 werden die beiden rechten Spalten der Matrix, womit jedes Symbol definiert wird, nicht abgebildet.
Durch Vergleich kann man den Unterschied selbst ermitteln.

```
10 SCREEN 0
20 PRINT CHR$(210)
```

und

```
10 SCREEN 1
20 PRINT CHR$(210)
```

Symbol
Code
32 &H20 33 &H21 34 &H22 35 &H23 36 &H24 37 &H25 38 &H26 39 &H27 40 &H28
Symbol
Code
41 &H29 42 &H2A 43 &H2B 44 &H2C 45 &H2D 46 &H2E 47 &H2F 48 &H30 49 &H31
Symbol
Code
50 &H32 51 &H33 52 &H34 53 &H35 54 &H36 55 &H37 56 &H38 57 &H39 58 &H3A
Symbol
Code
59 &H3B 60 &H3C 61 &H3D 62 &H3E 63 &H3F 64 &H40 65 &H41 66 &H42 67 &H43
Symbol
Code
68 &H44 69 &H45 70 &H46 71 &H47 72 &H48 73 &H49 74 &H4A 75 &H4B 76 &H4C
Symbol
Code
77 &H4D 78 &H4E 79 &H4F 80 &H50 81 &H51 82 &H52 83 &H53 84 &H54 85 &H55
Symbol
Code
86 &H56 87 &H57 88 &H58 89 &H59 90 &H5A 91 &H5B 92 &H5C 93 &H5D 94 &H5E
Symbol
Code
95 &H5F 96 &H60 97 &H61 98 &H62 99 &H63 100 &H64 101 &H65 102 &H66 103 &H67
Symbol
Code
104 &H68 105 &H69 106 &H6A 107 &H6B 108 &H6C 109 &H6D 110 &H6E 111 &H6F 112 &H70
Symbol
Code
113 &H71 114 &H72 115 &H73 116 &H74 117 &H75 118 &H76 119 &H77 120 &H78 121 &H79
Symbol
Code
122 &H7A 123 &H7B 124 &H7C 125 &H7D 126 &H7E 127 &H7F 128 &H80 129 &H81 130 &H82
Symbol
Code
131 &H83 132 &H84 133 &H85 134 &H86 135 &H87 136 &H88 137 &H89 138 &H8A 139 &H8B
Symbol
Code
140 &H8C 141 &H8D 142 &H8E 143 &H8F 144 &H90 145 &H91 146 &H92 147 &H93 148 &H94

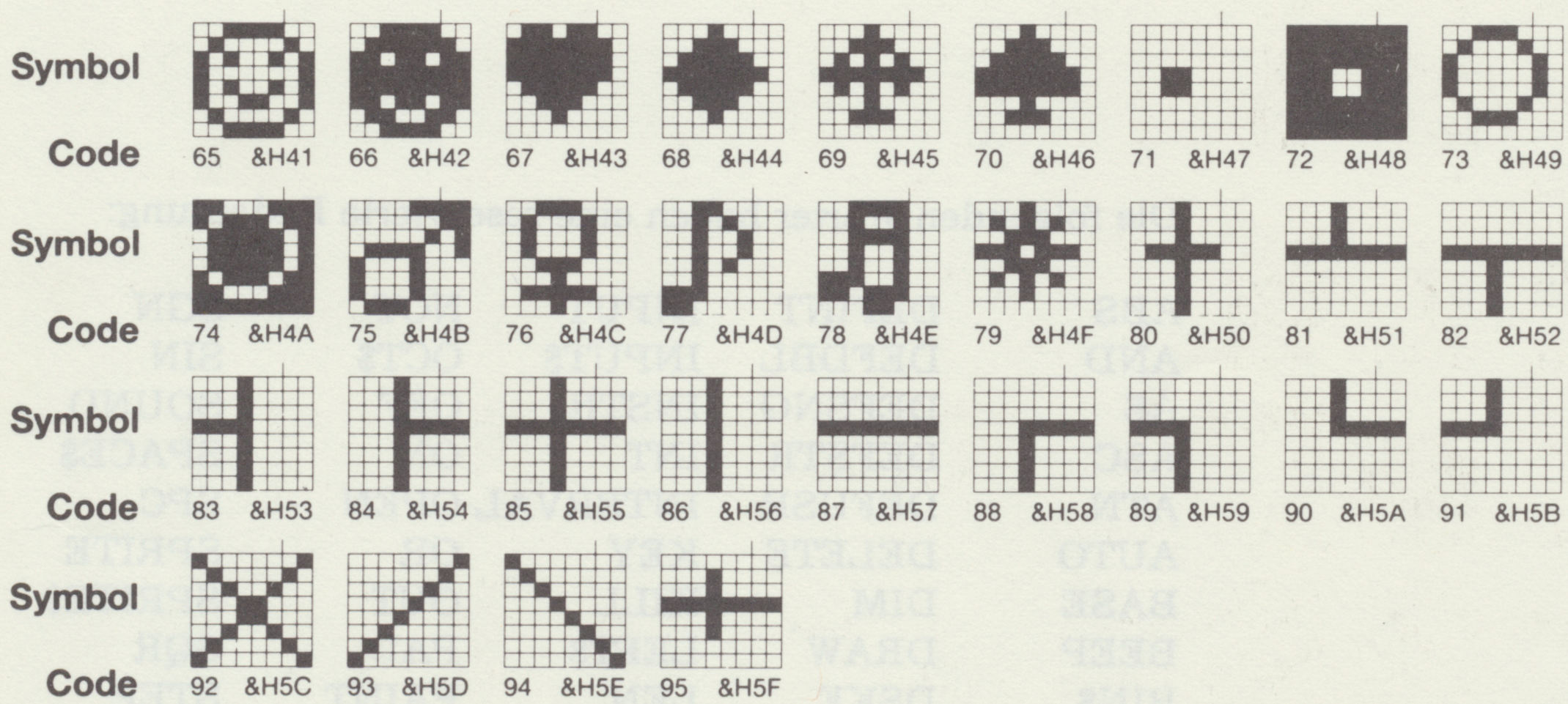
Symbol									
Code	149 &H95	150 &H96	151 &H97	152 &H98	153 &H99	154 &H9A	155 &H9B	156 &H9C	157 &H9D
Symbol									
Code	158 &H9E	159 &H9F	160 &HA0	161 &HA1	162 &HA2	163 &HA3	164 &HA4	165 &HA5	166 &HA6
Symbol									
Code	167 &HA7	168 &HA8	169 &HA9	170 &HAA	171 &HAB	172 &HAC	173 &HAD	174 &HAE	175 &HAF
Symbol									
Code	176 &HB0	177 &HB1	178 &HB2	179 &HB3	180 &HB4	181 &HB5	182 &HB6	183 &HB7	184 &HB8
Symbol									
Code	185 &HB9	186 &HBA	187 &HBB	188 &HBC	189 &HBD	190 &HBE	191 &HBF	192 &HC0	193 &HC1
Symbol									
Code	194 &HC2	195 &HC3	196 &HC4	197 &HC5	198 &HC6	199 &HC7	200 &HC8	201 &HC9	202 &HCA
Symbol									
Code	203 &HCB	204 &HCC	205 &HCD	206 &HCE	207 &HCF	208 &HD0	209 &HD1	210 &HD2	211 &HD3
Symbol									
Code	212 &HD4	213 &HD5	214 &HD6	215 &HD7	216 &HD8	217 &HD9	218 &HDA	219 &HDB	220 &HDC
Symbol									
Code	221 &HDD	222 &HDE	223 &HDF	224 &HE0	225 &HE1	226 &HE2	227 &HE3	228 &HE4	229 &HE5
Symbol									
Code	230 &HE6	231 &HE7	232 &HE8	233 &HE9	234 &HEA	235 &HEB	236 &HEC	237 &HED	238 &HEE
Symbol									
Code	239 &HEF	240 &HF0	241 &HF1	242 &HF2	243 &HF3	244 &HF4	245 &HF5	246 &HF6	247 &HF7
Symbol									
Code	248 &HF8	249 &HF9	250 &HFA	251 &HFB	252 &HFC	253 &HFD	254 &HFE	255 &HFF	

Alternative Symbole

Diese sind die Symbole, die man neben den bereits dargestellten Symbolen erhält, indem man CHR$(1) in die PRINT-Anweisung aufnimmt. Beispiel:

```
10 SCREEN 1
20 PRINT CHR$(65)
30 PRINT CHR$(1) + CHR$(65)
```

Zeile 20 generiert ein A und Zeile 30 hat zum Resultat, daß ein Gesicht abgebildet wird.



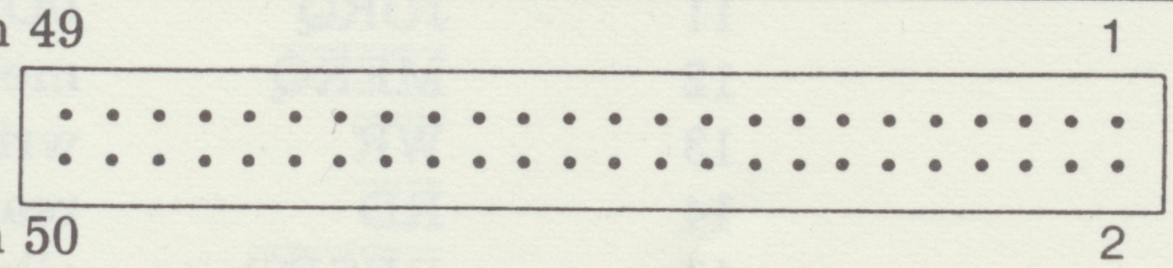
Anhang N Reservierte Namen

Die folgenden Wörter haben eine reservierte Bedeutung:

ABS	DEFINT	INPUT	NOT	SGN
AND	DEFDBL	INPUT$	OCT$	SIN
AS	DEFSNG	INSTR	OFF	SOUND
ASC	DEFSTR	INT	ON	SPACE$
ATN	DEFUSR	INTERVAL	OPEN	SPC
AUTO	DELETE	KEY	OR	SPRITE
BASE	DIM	KILL	OUT	SPRITE$
BEEP	DRAW	LEFT$	PAD	SQR
BIN$	DSKF	LEN	PAINT	STEP
BLOAD	ELSE	LET	PDL	STICK
BSAVE	END	LINE	PEEK	STOP
CALL	EOF	LIST	PLAY	STR$
CDBL	EQV	LLIST	POINT	STRIG
CHR$	ERASE	LOAD	POKE	STRING$
CINT	ERL	LOC	POS	SWAP
CIRCLE	ERR	LOCATE	PRINT	TAB
CLEAR	ERROR	LOF	PSET	TAN
CLOAD	EXP	LOG	PRESET	THEN
CLOSE	FIELD	LPOS	PUT	TIME
CLS	FILES	LPRINT	READ	TROFF
COLOR	FIX	LSET	REM	TRON
CONT	FN	MAXFILES	RENUM	USING
COPY	FOR	MERGE	RESTORE	USR
COS	FRE	MID$	RESUME	VAL
CSAVE	GET	MKD$	RETURN	VARPTR
CSNG	GOSUB	MKI$	RIGHT$	VDP
CSRLIN	GOTO	MOD	RND	VPEEK
CVD	HEX$	MOTOR	RSET	VPOKE
CVI	IF	MKS$	RUN	WAIT
CVS	IMP	NAME	SAVE	WIDTH
DATA	INKEY$	NEW	SCREEN	XOR
DEF	INP	NEXT		

Anhang O Technische Information

Anschluß Cartridgestecker

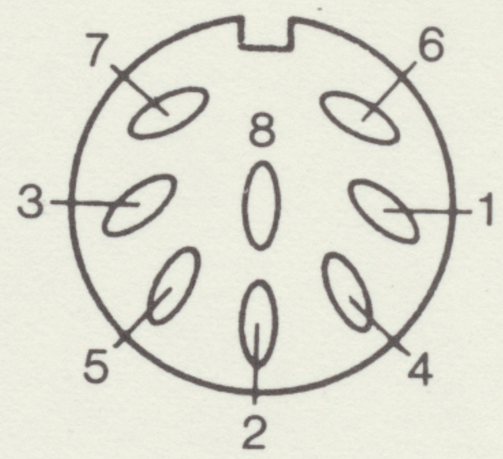


Pin	*Name*	*I/O*	*Pin*	*Name*	*I/O*
1	$\overline{CS1}$	O	2	$\overline{CS2}$	O
3	$\overline{CS12}$	O	4	$\overline{SLTSL}$	O
5	Reserve	–	6	$\overline{RFSH}$	O
7	$\overline{WAIT}$	I	8	$\overline{INT}$	I
9	$\overline{M1}$	O	10	$\overline{BUSDIR}$	I
11	$\overline{IORQ}$	O	12	$\overline{MERQ}$	O
13	$\overline{WR}$	O	14	$\overline{RD}$	O
15	$\overline{RESET}$	O	16	Reserve	–
17	A9	O	18	A15	O
19	A11	O	20	A10	O
21	A7	O	22	A6	O
23	A12	O	24	A8	O
25	A14	O	26	A13	O
27	A1	O	28	A0	O
29	A3	O	30	A2	O
31	A5	O	32	A4	O
33	D1	I/O	34	D0	I/O
35	D3	I/O	36	D2	I/O
37	D5	I/O	38	D4	I/O
39	D7	I/O	40	D6	I/O
41	GND	–	42	CLOCK	O
43	GND	–	44	SW1	–
45	+5V	–	46	SW2	–
47	+5V	–	48	+12V	–
49	SOUND-IN	I	50	–12V	–

Pin	*Name*	*Bedeutung*
1	$\overline{CS1}$	Auswahl ROM-Adressen 4000-7FFF
2	$\overline{CS2}$	Auswahl ROM-Adressen 8000-BFFF
3	$\overline{CS12}$	Auswahl ROM-Adressen 4000-BFFF (256kROM)
4	$\overline{SLTSL}$	Slot-Auswahl
5	Reserve	
6	$\overline{RFSH}$	Refresh-Zyklus
7	$\overline{WAIT}$	WAIT-request-Signal von CPU
8	$\overline{INT}$	Interrupt-request von CPU
9	$\overline{M1}$	fetch-Zyklus von CPU
10	$\overline{BUSDIR}$	Steuert externen Databus Puffer
11	$\overline{IORQ}$	I/O request Signal
12	$\overline{MERQ}$	memory (Speicher) request
13	$\overline{WR}$	write (schreiben)
14	$\overline{RD}$	read (lesen)
15	$\overline{RESET}$	allgemein reset (Rückschaltung)
16	Reserve	
17-32	A0-A15	Adressen-Leitungen
33-40	D0-D7	Daten-Leitungen
41	GND	Masse
42	CLOCK	3.579545 MHz CPU Takt
43	GND	Masse
44,46	SW1,SW2	Sicherung Ein- und Ausnehmen
45,47	+5V	Versorgungsspannung
48	+12V	Versorgungsspannung
49	SOUNDIN	Input Ton
50	-12V	Versorgungsspannung

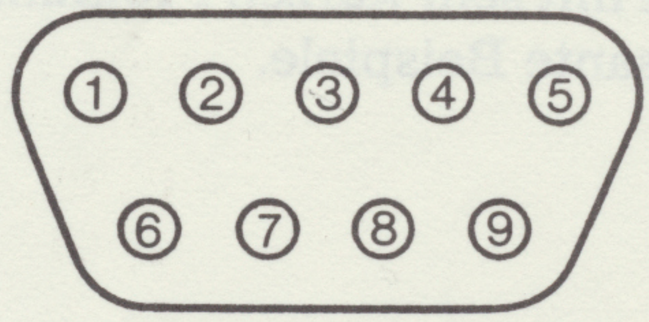
Anschluß Kassettenrecorder

Pin	*Name*	*I/O*
1	GND	
2	GND	
3	GND	
4	CMTOUT	O
5	CMTIN	I
6	REM +	O
7	REM –	O
8	GND	



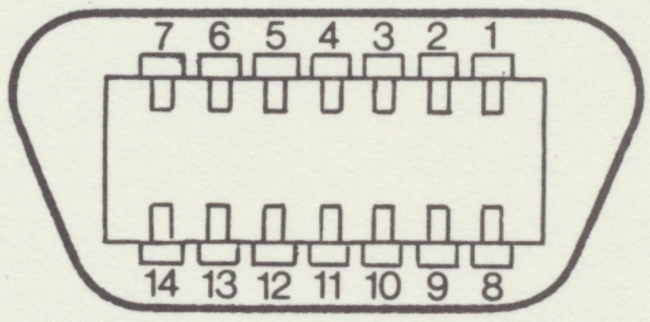
Anschluß Joystick

Pin	*Name*	*I/O*
1	FWD	I
2	BACK	I
3	LEFT	I
4	RIGHT	I
5	+ 5 V	
6	TRG 1	I/O
7	TRG 2	O
8	Output	O
9	GND	



Anschluß Drucker

Pin	*Name*	*(sofern vorhanden)*
1	PSTB	
2	PDB0	
3	PDB1	
4	PDB2	
5	PDB3	
6	PDB4	
7	PDB5	
8	PDB6	
9	PDB7	
10	NC	
22	BUSY	
12	NC	
13	NC	
14	GND	

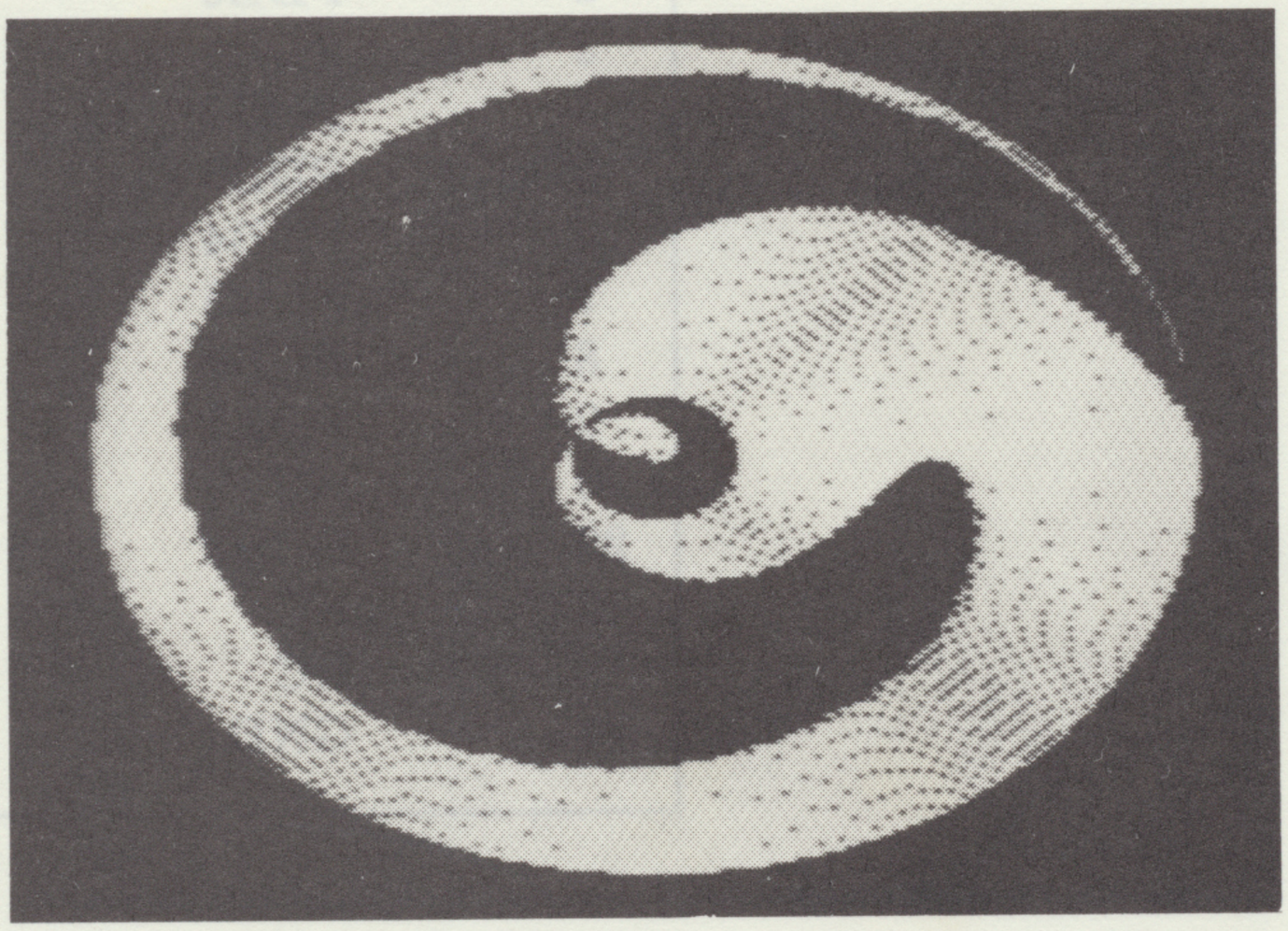


Programme

1. Grafisches Potpourri

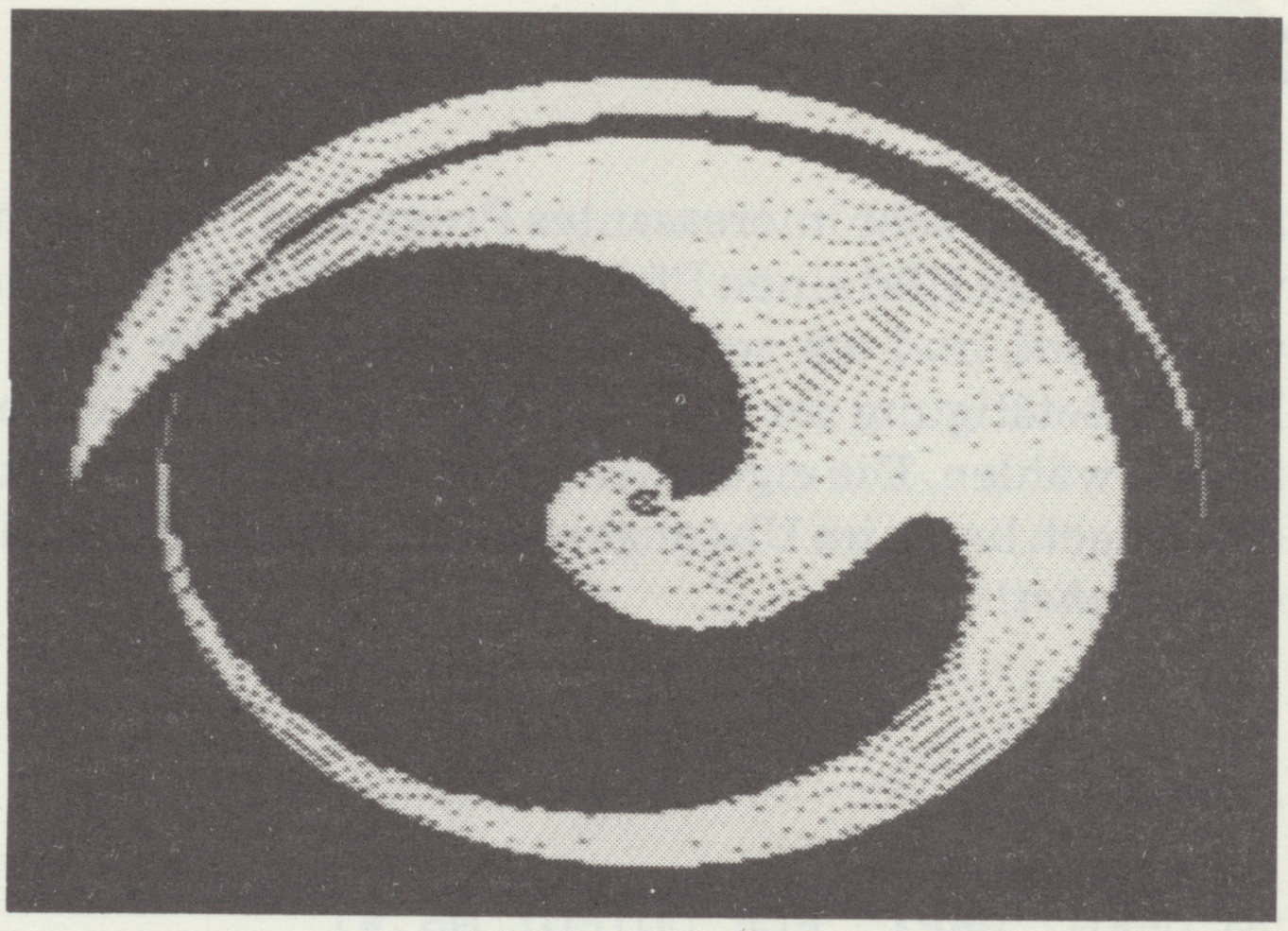
Wir zeigten bereits zum Abschluß von Kapitel 16 einige Beispiele hübscher Darstellungen, die mit sehr kurzen Programmen erzielt werden können. In der Folge einige weitere interessante Beispiele.

```
10 REM SCHLANGE
20 PI=3.14159
30 SCREEN 2
40 FOR R=1 TO 80
50 CIRCLE(128,96),80-R,15,SIN(R/80*PI)*2*PI,R/80*PI
60 NEXT R
70 GOTO 70
```



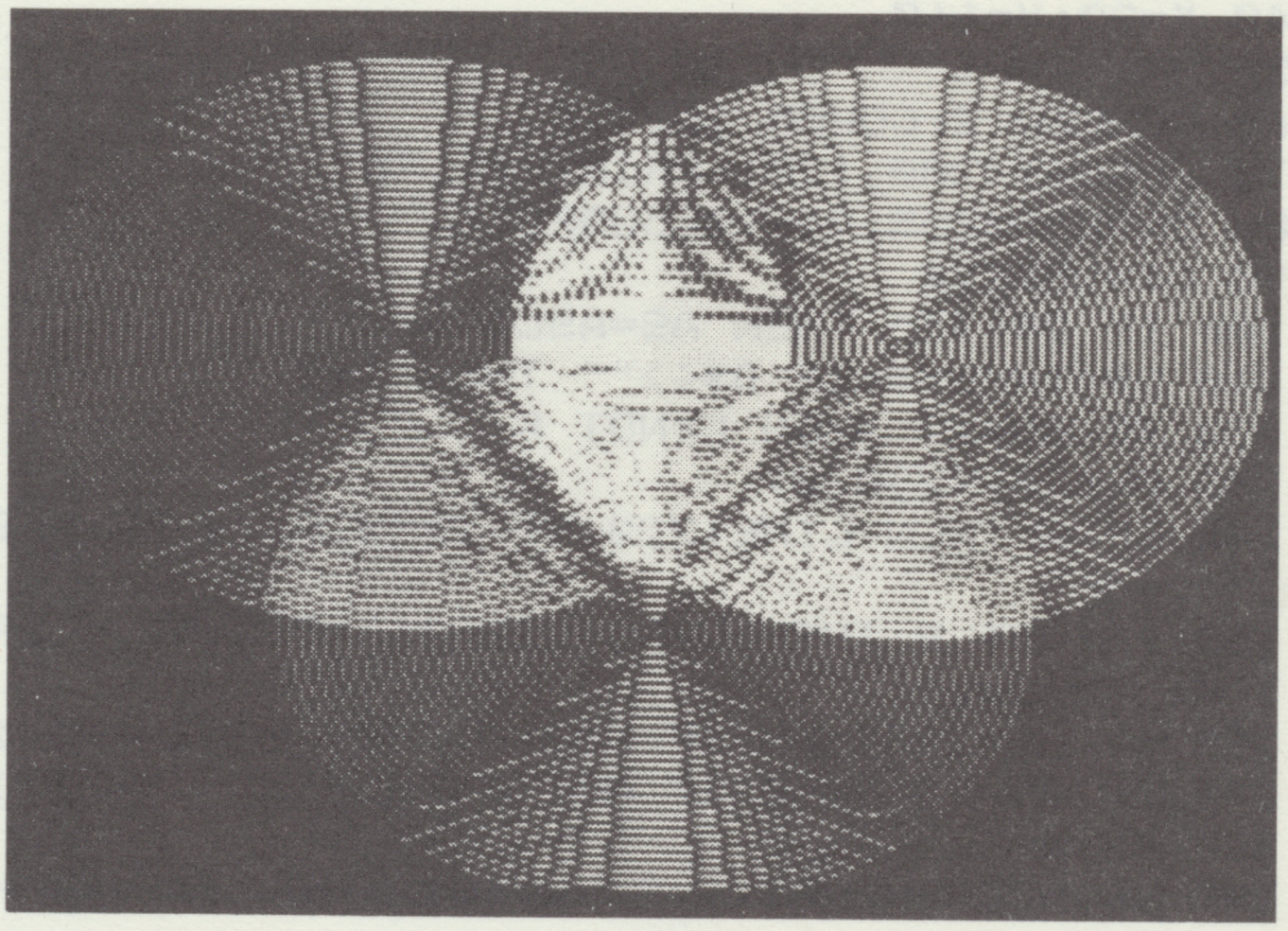
Schlange

```
10 REM VOGEL
20 PI=3.14159
30 SCREEN 2
40 FOR R=1 TO 80
50 CIRCLE(128,96),R,15,
   SIN(R/80*PI)*2*PI,R/80*PI
60 NEXT R
70 GOTO 70
```



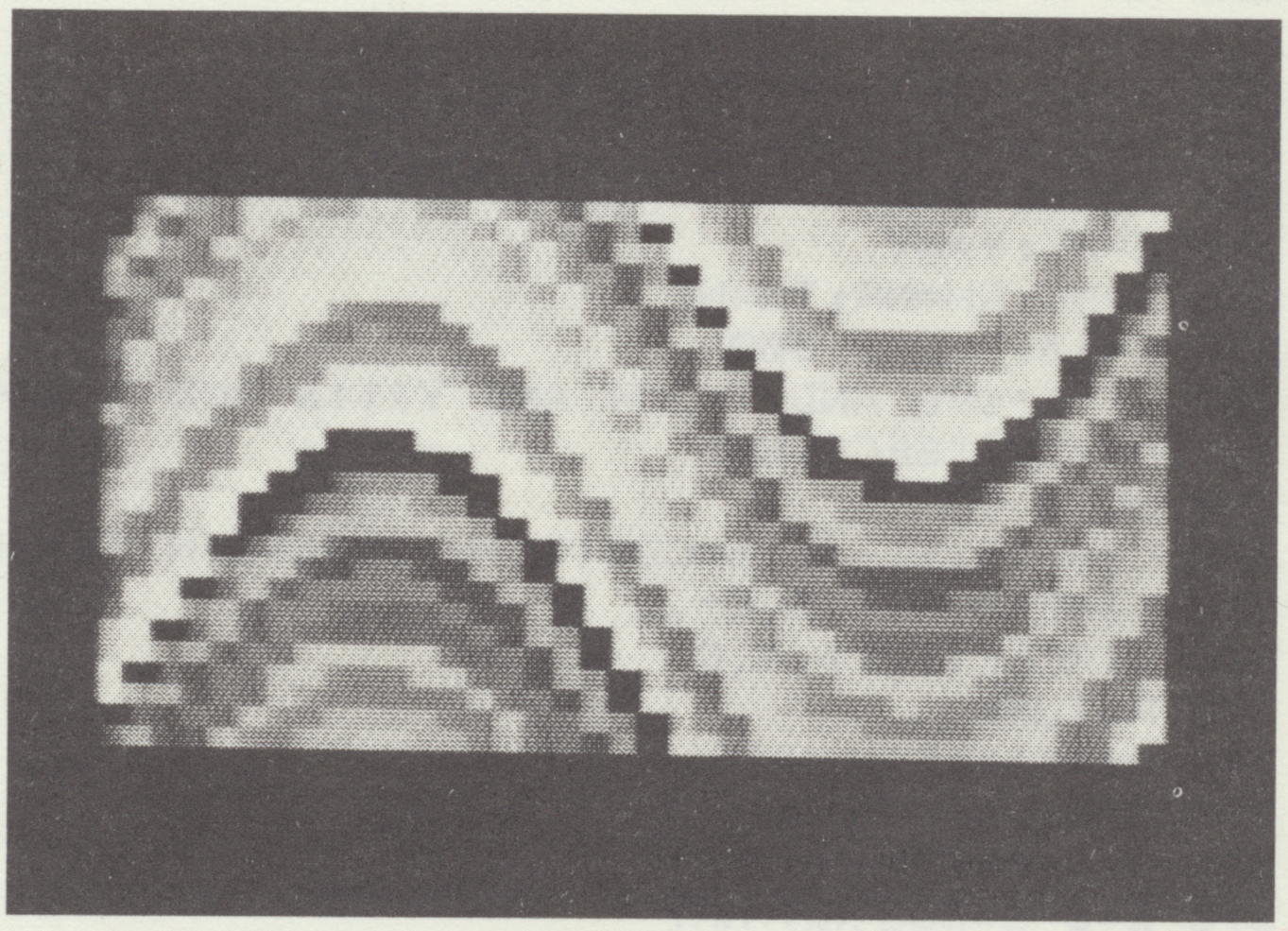
Vogel

```
10 REM 3 KREISE
20 SCREEN 2
30 FOR R=1 TO 65 STEP 2
40 CIRCLE(85,65),R,2
50 CIRCLE(170,65),R,7
60 CIRCLE(128,130),R,8
70 NEXT R
80 GOTO 80
```



Drei Kreise

```
10 REM FARBENSINUS
20 SCREEN 3
30 PI=3.14159
40 FOR X=50 TO 200 STEP 4
50 FOR Y=45 TO 145 STEP 4
60 PSET(X,Y),((Y/6.25+SIN((X-50)
   /75*PI)*8+8)MOD 15+1)
70 NEXT Y
80 NEXT X
90 GOTO 90
```



Farbensinus

2. Segeln

Segeln ist ein sehr interessantes Programm. Nach dem Eingeben des RUN- Befehls sehen wir rechts im Bild einen Pfeil. Hiermit wird die Windrichtung angegeben. Links sehen wir unser Boot, und wir sehen außerdem noch die Boje, die angesegelt werden muß. Durch Betätigung der ESC-Taste können wir nacheinander den Stand des Ruders und des Segels wählen. Die eigentliche Einstellung der Position geschieht mit den Cursor-Tasten (Pfeil nach links: im Uhrzeigersinn; Pfeil nach rechts: in Gegenrichtung des Uhrzeigers).
Gute Fahrt!

Programm

```
10 TIME=0
20 OPEN "GRP:" FOR OUTPUT AS #1
30 PI=3.14159265#
40 X=20:Y=160
50 B=1:H=PI
60 SCREEN 2:COLOR 15,4,4
70 PRESET(200,120):PRINT #1,"O"
80 DRAW"BM255,91L10E5G5F5"
90 Q$=INKEY$
100 IF Q$<>"" THEN AS=ASC(Q$)
110 D$=INKEY$:IF D$<>"" THEN 110
120 IF AS=27 THEN R=NOT R
130 PRESET(230,0)
140 IF R THEN LINE(220,0)-(255,8),4,BF:PRESET(220,0):PRINT #1
,"Ruder"
150 IF NOT R THEN LINE(220,0)-(255,8),4,BF:PRESET(220,0):PRIN
T #1,"Segel"
160 IF AS=29 THEN IF NOT R THEN NZ=ZH+PI/16 ELSE NR=RH+PI/50
170 IF AS=28 THEN IF NOT R THEN NZ=ZH-PI/16 ELSE NR=RH-PI/50
180 AS=0
190 V=V+COS(NZ-PI/2+PI-H)*SIN(NZ-PI/2+PI-H)-ABS(V/10)^1.5*SGN
(V)
200 NH=H-SIN(NR)*ABS(V)
210 NX=X-ABS(V)*SIN(PI-NH)
220 NY=Y-ABS(V)*COS(PI-NH)
230 C=4:GOSUB 300
240 GOSUB 340
250 X=NX:Y=NY
260 H=NH:RH=NR:ZH=NZ
270 C=15:GOSUB 300
280 IF X>196 AND X<204 AND Y>116 AND Y<124 THEN SCREEN 0:PRIN
T TIME/50;"SEC":END
290 GOTO 90
300 LINE(X,Y)-(X+6*COS(ZH+H),Y+6*SIN(ZH+H)),C
310 LINE(A,B)-(D,E),C
320 CIRCLE(X,Y),4,C
330 RETURN
340 A=NX+4*SIN(NH)
350 B=NY-4*COS(NH)
360 D=A+6*SIN(NH+NR)
370 E=B-6*COS(NH+NR)
380 RETURN
```

3. Integrieren

Mit nachstehendem Programm ist das Integral einer Funktion zwischen zwei Grenzen zu berechnen.
Als erstes wird die Anzahl der Schritte erfragt, durch die das Integral berechnet werden soll. Im Programm wird Gebrauch gemacht vom Gesetz von Simpson.
Die Funktion ist in einer Subroutine untergebracht: Zeile 220-230. In diesem Fall handelt es sich als Beispiel um die dritte Potenz von X.

Programm

```
10 CLS
20 INPUT"ANZAHL SCHRITTE: ";N
30 INPUT"UNTERE GRENZE:   ";A
40 INPUT"OBERE GRENZE:    ";B
50 H=(B-A)/N
60 FOR K=1 TO N-1
70 X=A+K*H
80 C=Q*2+4
90 Q=NOT(Q)
100 GOSUB 220
110 SUMME=SUMME+C*Y
120 NEXT K
130 X=A
140 GOSUB 220
150 FA=Y
160 X=B
170 GOSUB 220
180 FB=Y
190 I=H/3*(FA+SOMME+FB)
200 PRINT "INTEGRATION ERGIBT: ";I
210 END
220 Y=X*X*X
230 RETURN
```

Beispiel

Anzahl Schritte: 20
Untere Grenze: 0
Obere Grenze: 2
Integration ergibt: 4

4. Regression und Korrelation

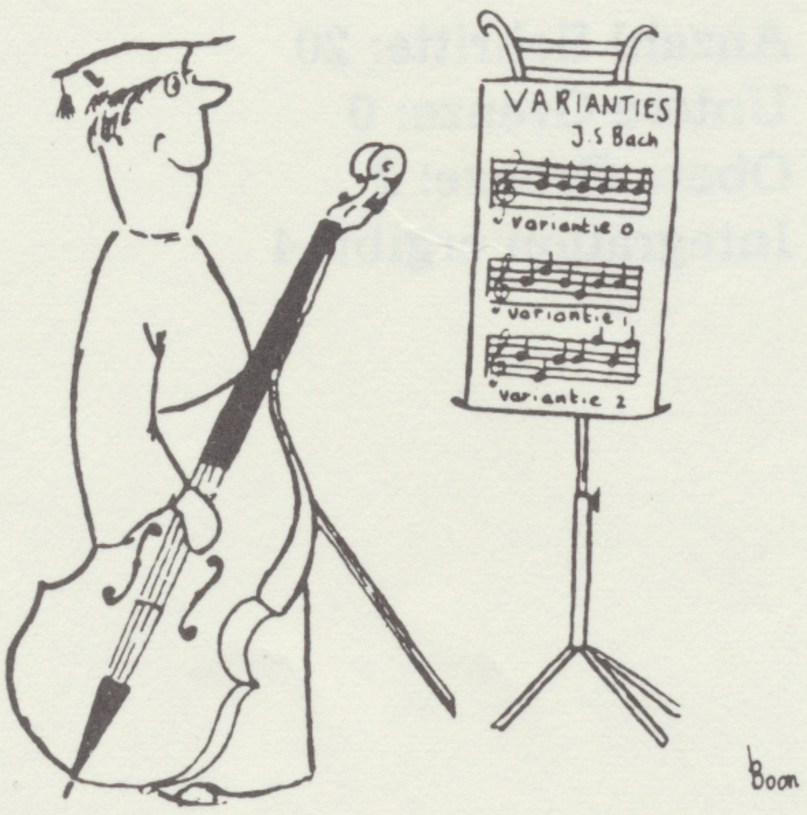
Die Regressions- und Korrelationsberechnung befaßt sich mit der Frage, wie man durch eine gegebene Anzahl von Punkten so gut wie möglich eine Gerade ziehen kann. Diese Gerade wird durch zwei Größen bestimmt: den Neigungswinkel M und den Punkt, in dem die Linie die Y-Achse schneidet (B). Jeder Punkt wird bestimmt durch eine X- und eine Y- Koordinate. Das Programm erfragt als erstes, wieviele Punkte eingegeben werden müssen.
Nach Eingabe der Koordinaten werden dann die Werte M und B berechnet. Schließlich wird der Korrelationskoeffizient R gezeigt. Wenn alle Punkte genau auf einer Geraden liegen, wird der absolute Wert von R gleich 1 sein. Liegen die Punkte nicht genau auf einer Geraden, so ist der absolute Wert von R weniger als 1. Offenbar gibt der Wert von R an, wie genau die Punkte wohl auf der berechneten Geraden liegen.

Programm

```
10 CLS
20 INPUT "Anzahl Punkte: ";N
30 FOR K=1 TO N
40 PRINT "X";K;
50 INPUT"=";X
60 PRINT "Y";K;
70 INPUT"=";Y
80 XS=XS+X
90 YS=YS+Y
100 X2=X2+X*X
110 Y2=Y2+Y*Y
120 XY=XY+X*Y
130 NEXT K
140 M=(XS*YS/N-XY)/(XS*XS/N-X2)
150 PRINT "M=";M
160 PRINT "B=";YS/N-M*XS/N
170 NM=SQR((N*X2-XS*XS)*(N*Y2-YS*YS))
180 T=N*XY-XS*YS
190 PRINT "R=";T/NM
```

Beispiel

Anzahl Punkte: 2
X1 = 1
Y1 = 2
X2 = 2
Y2 = 2
M = 2
B = 0
R = 1



5. Magnet

Wieder ein originelles Programm. Jedermann weiß, daß ein Magnet einen anderen Magneten wegdrücken kann, weil die gleichen Pole einander abstoßen. Auf diesem schon lange bekannten naturwissenschaftlichen Phänomen basiert das folgende Programm.
Ihren Magneten können Sie mit den Cursor-Tasten steuern. Versuchen Sie, in möglichst kurzer Zeit das angegebene Ziel zu erreichen.

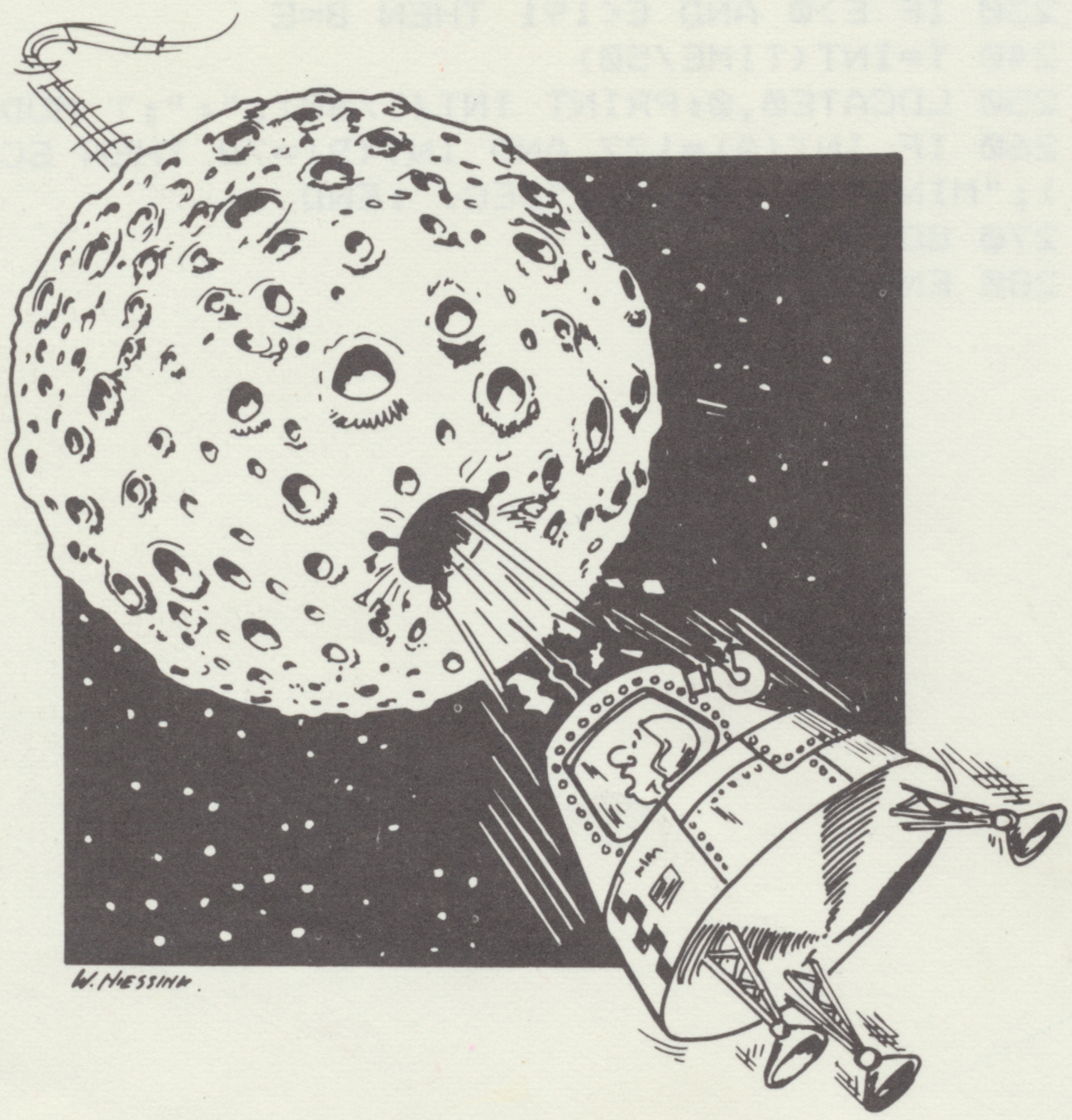
Programm

```
10 SCREEN 1,0
20 FOR T=0 TO 32:VPOKE6880+T,0:NEXT T
30 A=20:B=10
40 X=5:Y=5
50 TIME=0
60 SPRITE$(0)=CHR$(&H7E)+CHR$(&H42)+CHR$(&H81)+CHR$(&H81)+CHR
$(&H81)+CHR$(&H81)+CHR$(&H42)+CHR$(&H7E)
70 SPRITE$(1)=CHR$(&H18)+CHR$(&H7E)+CHR$(&H7E)+CHR$(&HFF)+CHR
$(&HFF)+CHR$(&H7E)+CHR$(&H7E)+CHR$(&H18)
80 VPOKE6480,15
90 Q$=INKEY$
100 IF Q$<>"" THEN AS=ASC(Q$)
110 IF AS=30 THEN Y=Y-3
120 IF AS=31 THEN Y=Y+3
130 IF AS=28 THEN X=X+3
140 IF AS=29 THEN X=X-3
150 AS=0
160 PUT SPRITE1,(X,Y),15,1
170 PUT SPRITE0,(A,B),15,0
180 D=SQR((A-X)^2+(B-Y)^2)
190 F=1/D
200 C=A-COS((A-X)/D)*F*20*SGN(X-A)
210 E=B+SIN((B-Y)/D)*F*20
220 IF C>0 AND C<255 THEN A=C
230 IF E>0 AND E<191 THEN B=E
240 T=INT(TIME/50)
250 LOCATE0,0:PRINT INT(T/60);":";T MOD 60
260 IF INT(A)=127 AND INT(B)=78 THEN SCREEN 0:PRINT INT (T/60
);"MIN.";T MOD 60;"SEC.":END
270 GOTO 90
280 END
```

6. Jupiterlandung

Das folgende Programm zeigt uns unser Weltraumschiff, das wir mit Hilfe der Cursor-Tasten sicher auf dem Boden des Jupiter landen lassen sollen.
Gelingt es uns - oder ist Jupiter die absolute Endstation bei einer Bruchlandung?

```
10 SCREEN 1
20 FOR T=0 TO 32:VPOKE6880+T,194:NEXT T
30 V=-20
40 S=180
50 TA=200
60 BR=1
70 SPRITE$(1)=CHR$(&H8)+CHR$(&H1C)+CHR$(&H3E)+CHR$(&H3E)+CHR$
(&H3E)+CHR$(&H3E)+CHR$(&H7F)+CHR$(&H7F)
80 Q$=INKEY$:IF Q$<>"" THEN AS=ASC(Q$)
90 IF AS=30 THEN BR=BR+.1
100 IF AS=31 AND BR>0 THEN BR=BR-.1
110 AS=0
120 FOR T=0 TO 100:NEXT T
130 TA=TA-BR
140 A=BR-1
150 V=V+A
160 S=S+(V-A/2)/10
170 IF TA>0 THEN LOCATE11,0:PRINT"      ":LOCATE 0,0:PRINT "Re
servoir:";TA
180 LOCATE 19,1:PRINT"      "
190 LOCATE 0,1:PRINT "Raketenschub: ";BR
200 LOCATE10,2:PRINT"        "
210 LOCATE 0,2:PRINT "Geschwindigkeit: ";V
220 PUT SPRITE 0,(120,175-S),15,1
230 IF ABS(V)<.5 AND ABS(S)<.5 THEN LOCATE 0,3:PRINT"Gelungen
":END
240 IF TA<=0 THEN LOCATE 0,0:PRINT "Tank leer":BR=0
250 IF S>0 THEN 80
260 PUT SPRITE 0,(120,175-S),0,1
270 P=219:GOSUB 1000
280 P=215:GOSUB 1000
290 P= 15:GOSUB 1000
300 P=  7:GOSUB 1000
310 P= 46:GOSUB 1000
320 P= 32:GOSUB 1000
330 END
1000 VPOKE6860,P
1010 VPOKE6861,P
1020 VPOKE6862,P
1030 VPOKE6863,P
1040 VPOKE6864,P
1050 VPOKE6865,P
1060 VPOKE6866,P
1070 VPOKE6867,P
1080 VPOKE6829,P
1090 VPOKE6830,P
1100 VPOKE6831,P
1110 VPOKE6832,P
1120 VPOKE6833,P
1130 VPOKE6834,P
1140 FOR T=0 TO 10:NEXT T
1150 RETURN
```



7. Jahrhundertkalender

Das folgende Programm bestimmt für jedes Datum dieses und des vorigen Jahrhunderts den entsprechenden Wochentag.
Damit können Sie endlich feststellen, an welchem Tag Sie geboren sind. Das nachstehende Beispiel bezieht sich auf den 30. August 1981.

Programm

```
5 CLS
10 INPUT"Jahr: ";J
20 INPUT"Monat: ";M
30 INPUT"Tag: ";D
40 IF M<=2 THEN J=J-1:M=M+12
50 E=INT((13/5)*(M+1))
60 F=INT(5*J/4)
70 G=INT(J/100)
80 H=INT(J/400)
90 T=D+E+F-G+H
100 DA=1+INT((T/7-INT(T/7))*7+.1)
110 ON DA GOTO 120,121,122,123,124,125,126
120 PRINT "Samstag":GOTO 130
121 PRINT "Sonntag":GOTO 130
122 PRINT "Montag":GOTO 130
123 PRINT "Dienstag":GOTO 130
124 PRINT "Mittwoch":GOTO 130
125 PRINT "Donnerstag":GOTO 130
126 PRINT "Freitag"
130 END
```

Beispiel

Jahr: 1981
Monat: 8
Tag: 30
Sonntag

8. Marsmännchen

Brrr...an manchen Stellen im Weltraum können viele Marsmännchen vorkommen. Aufgepaßt vor Zusammenstößen mit diesen grinsenden Wesen.
Als Kapitän Ihres Raumschiffes beherrschen Sie natürlich die Cursor- Tasten mit Fingerspitzengefühl. Aber wie lange können Sie dieser Marsmännchenflut ausweichen?

Programm

```
10 TIME=0
20 SCREEN 1
30 X=128
40 FOR T=0 TO 32:VPOKE(6880+T),0:NEXT T
50 SPRITE$(0)=CHR$(&H81)+CHR$(&H42)+CHR$(&H18)+CHR$(&H3C)+CHR
$(&H66)+CHR$(&HC3)+CHR$(&H81)+CHR$(&H18)
60 Q$=INKEY$
70 IF Q$<>""THEN AS=ASC(Q$)
80 IF AS=28 AND X<246 THEN X=X+2
90 IF AS=29 AND X>16  THEN X=X-2
100 AS=0
110 PUT SPRITE 0,(X,0),15,0
120 IF VPEEK(6144+INT(X/8))=1 OR VPEEK(6144+INT(X/8+.85))=1 T
HEN BEEP:P=P+1
130 IF P=10 THEN 230
140 R1=30*RND(1)+1
150 R2=30*RND(1)+1
160 C1=INT(1.3*RND(1))
170 C2=INT(1.3*RND(1))
180 VPOKE 6784+R1,1+C1
190 VPOKE 6784+R2,1+C2
200 LOCATE 0,23:PRINT
210 VPOKE 6888,48+P
220 GOTO 60
230 SCREEN 0
240 PRINT TIME/50;"SEC"
```

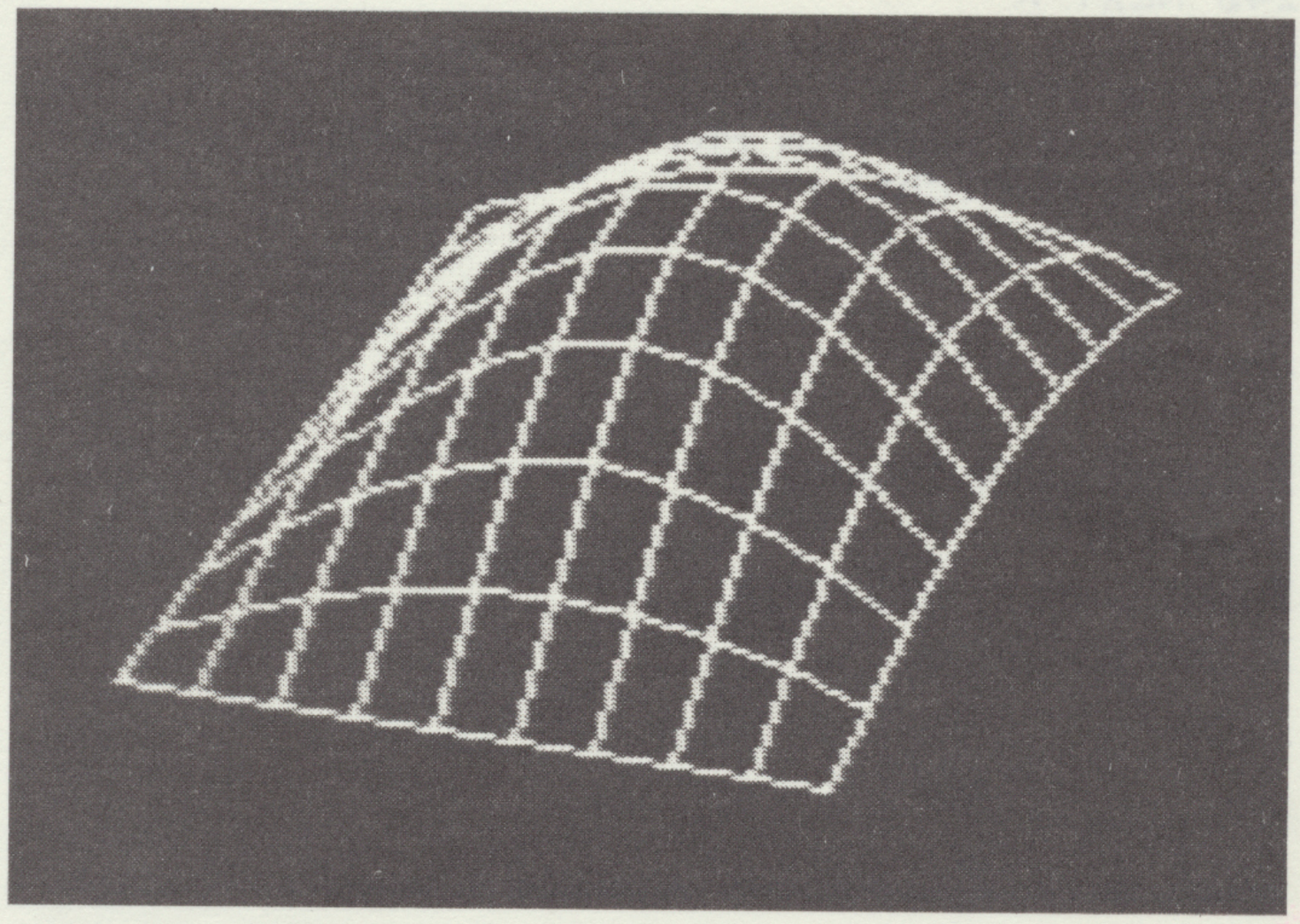
9. 3D

Der Ausdruck '3D' bedeutet 'dreidimensional' oder, anders ausgedrückt, 'räumlich'. Mit unserem MSX-Computer können wir allerhand schöne räumliche Darstellungen machen. Nachstehendes Programm zeigt ein einfaches, aber gewiß sehr eindrucksvolles Beispiel.

Programm

```
10 PI=3.14159265#
20 DIM MX(10,10),MY(10,10)
30 CLS
40 FOR X=0 TO 9
50 FOR Y=0 TO 9
60 GOSUB 240
70 ZX=50+X*10+Y*5
80 ZY=120-Y*9-Z+X*2
90 MX(X,Y)=ZX
100 MY(X,Y)=ZY
110 LOCATE 0,0:PRINT X;Y
120 NEXT Y
130 NEXT X
140 SCREEN 2
150 FOR X=0 TO 8
160 FOR Y=0 TO 8
170 LINE(MX(X,Y),MY(X,Y))-(MX(X+1,Y),MY(X+1,Y))
180 LINE(MX(X,Y),MY(X,Y))-(MX(X,Y+1),MY(X,Y+1))
190 NEXT Y
200 LINE(MX(X,9),MY(X,9))-(MX(X+1,9),MY(X+1,9))
210 LINE(MX(9,X),MY(9,X))-(MX(9,X+1),MY(9,X+1))
220 NEXT X
230 GOTO 230
240 Z=50*(SIN(X/10*PI)*SIN(Y/10*PI))
250 RETURN
```

Resultat



10. Wer ist am schlausten?

Dieses Programm gibt ein typisches Buchstabenspiel. Der Computer vermittelt jeweils eine Anzahl Buchstaben, und Sie müssen raten, um was für ein Wort es sich handelt. Dies ist vor allem ein Spiel, das die Jüngeren unter uns besonders spannend finden. Die Älteren können dagegen als Übung probieren, den Aufbau dieses Programmes zu ermitteln.
Schließlich noch ein Hinweis: Bei diesem Spiel zuerst die CAPS-Taste eindrücken: also nur Großbuchstaben verwenden.

Programm

```
10 PRINT"Mit Tastendruck Spiel beginnen"
20 W=RND(1):IF INKEY$="" THEN 20
30 DIM A$(10,5)
40 DIM B$(5)
50 G$=""
60 P$="THCEHLESMAREGITARBOKETSEFGINOHEGNULEKNERKNURTTDATS"
70 FOR K=1 TO 10
80 FOR L=1 TO 5
90 B1=(K-1)*5
100 A$(K,L)=MID$(P$,B1+L,1)
110 NEXT L
120 NEXT K
130 R=INT(RND(1)*10+1)
140 FOR K=5 TO 1 STEP -1
150 B$(K)=A$(R,K)
160 G$=G$+B$(K)
170 NEXT K
180 X$=""
190 R=INT(RND(5)*5+1)
200 FOR K=R TO R+4
210 L=K
220 IF K>5 THEN L=L-5
230 X$=X$+B$(L)
240 NEXT K
250 PRINT X$;" ist  ";
260 INPUT M$
270 IF M$<>G$ THEN PRINT "Falsch":GOTO 250
280 PRINT"Richtig"
```

Beispiel

```
KARBO heißt: ?RAKBO
FALSCH
KARBO heißt: ?KOBRA
RICHTIG
```

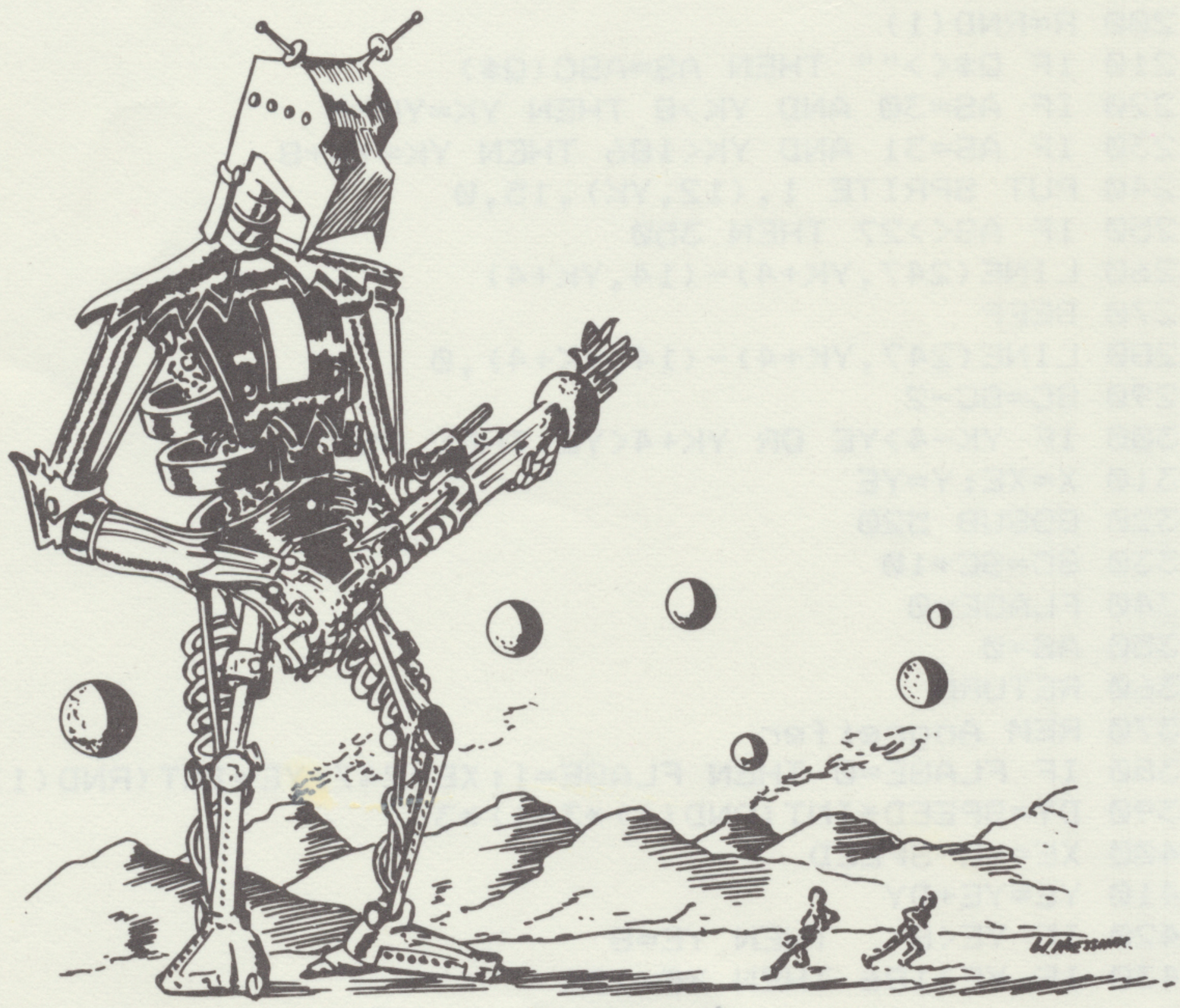
11. Merkurungeheuer

Schon wieder droht Gefahr: Merkurungeheuer im Anflug! Sie sind der mutige Pilot in einem Jäger, und mit dem Finger an der ESC-Taste müssen Sie die Gefahr bannen. Sie können schließlich diesen schrecklichen Ungeheuern ausweichen, indem Sie im rechten Augenblick die Cursor-Tasten betätigen. Aufgepaßt, die Ungeheuer werden zunehmend aggressiver!

Programm

```
10 OPEN "GRP:" FOR OUTPUT AS #1
20 SCREEN 2:COLOR 15,4,4
30 SPRITE$(0)=CHR$(&H10)+CHR$(&H30)+CHR$(&HF0)+CHR$(&HFF)+CHR
$(&HFF)+CHR$(&HF0)+CHR$(&H30)+CHR$(&H10)
40 SPRITE$(1)=CHR$(&H6)+CHR$(&HC)+CHR$(&H1C)+CHR$(&HFF)+CHR$(
&HFF)+CHR$(&H1C)+CHR$(&HC)+CHR$(&H6)
50 YK=96
60 SPEED=0
70 XE=246:YE=160
80 REM HAUPTSCHLEIFE
90 GOSUB 180
100 N=N+1
110 IF N>999 THEN GOSUB 640
120 GOSUB 370
130 IF SC> 60 THEN FIRE=0:SPEED=2
140 IF SC>100 THEN FIRE=-1:SPEED=1
150 IF SC>150 THEN FIRE=-1:SPEED=2
160 IF SC>200 THEN GOSUB 630
170 GOTO 90
180 IF HIT<>0 THEN HIT=HIT-1:RETURN
190 Q$=INKEY$
200 R=RND(1)
210 IF Q$<>"" THEN AS=ASC(Q$)
220 IF AS=30 AND YK>8 THEN YK=YK-8
230 IF AS=31 AND YK<186 THEN YK=YK+8
240 PUT SPRITE 1,(12,YK),15,0
250 IF AS<>27 THEN 350
260 LINE(247,YK+4)-(14,YK+4)
270 BEEP
280 LINE(247,YK+4)-(14,YK+4),0
290 SC=SC-2
300 IF YK-4>YE OR YK+4<YE THEN 350
310 X=XE:Y=YE
320 GOSUB 520
330 SC=SC+10
340 FLAGE=0
350 AS=0
360 RETURN
370 REM Angreifer
380 IF FLAGE=0 THEN FLAGE=1:XE=247:YE=INT(RND(1)*22+2)*8
390 DY=SPEED*INT(RND(1)*3-1)*3
400 XE=XE-SPEED
410 YE=YE+DY
420 IF YE<8   THEN YE=8
430 IF YE>186 THEN YE=186
```

```
440 IF XE<10  THEN FLAGE=0:GOTO 500
450 PUT SPRITE 0,(XE,YE),15,1
460 IF NOT FIRE OR RND(1)>.15 THEN 500
470 LINE(10,YE+4)-(XE,YE+4)
480 LINE(10,YE+4)-(XE,YE+4),0
490 IF YE-3<YK AND YE+3>YK THEN HIT=5:X=14:Y=YK:GOSUB 520
500 RETURN
510 REM Explosion
520 PRESET(X,Y):PRINT #1,CHR$(1);CHR$(71)
530 PUT SPRITE 0,(0,0),0,1
540 PRESET(X,Y):PRINT #1,CHR$(249)
550 PRESET(X,Y):PRINT #1,CHR$(248)
560 PRESET(X,Y):PRINT #1,CHR$(1);CHR$(79)
570 PRESET(X,Y):PRINT #1,CHR$(1);CHR$(73)
580 PRESET(X,Y):PRINT #1,CHR$(215)
590 PRESET(X,Y):PRINT #1,CHR$(219)
600 FOR T=0 TO 20:NEXT T
610 LINE(X,Y)-(X+8,Y+8),0,BF
620 RETURN
630 SCREEN 0:PRINT"Sie haben es geschafft, den Feind zu verj
agen, herzlichen Glückwunsch.    Sie hatten ";N/100;" Lichts
ekunden nötig.":END
640 SCREEN 0:PRINT"Als Jäger können wir Sie nicht längergebr
auchen, und Sie werden deshalb   degradiert.":END
```



12. B-52 Gänse

Die Gänseart, die mit dem Codenamen B-52 gekennzeichnet wird, ist eine höchst gefährliche Sorte. Es handelt sich nicht um ein reines Naturprodukt. Diese Gänse können schließlich richtige Bombenangriffe fliegen, und welche echte Gans tut sowas?
Es ist wohl klar, daß solche Gänse abgeschossen werden müssen. Ihre Kanone können Sie mit den Cursor-Tasten richten und mit der Leertaste abfeuern. Hüten Sie sich aber davor, daß die Gänse Ihre Kanone treffen. Dann können Sie zwar noch immer feuern, aber Ihre Bewegungsfreiheit haben Sie verloren. Wie gut, daß wir diese B-52 Gänse nicht allzu oft zu sehen bekommen!

Programm

```
10 SCREEN 2
20 COLOR 15,4,14
30 S$=CHR$(&H0)+CHR$(&H0)+CHR$(&H0)+CHR$(&HE)+CHR$(&HFF)
40 SPRITE$(1)=CHR$(&H0)+CHR$(&H4)+CHR$(&HC)+CHR$(&H1F)+CHR$(&
HFE)
50 SPRITE$(2)=S$
60 SPRITE$(3)=CHR$(&H0)+CHR$(&H0)+CHR$(&H0)+CHR$(&H0)+CHR$(&H
FF)+CHR$(&H1C)+CHR$(&HC)+CHR$(&H8)
70 SPRITE$(4)=S$
80 SPRITE$(0)=CHR$(&H8)
90 SPRITE$(5)=CHR$(&H8)+CHR$(&H8)+CHR$(&H3E)+CHR$(&H3E)+CHR$(
&H7F)+CHR$(&H7F)+CHR$(&H41)+CHR$(&H22)
100 SPRITE$(6)=CHR$(&H0)+CHR$(&H0)+CHR$(&H0)+CHR$(&H24)+CHR$(
&H24)+CHR$(&H66)+CHR$(&H7E)+CHR$(&HFF)
110 SPRITE$(7)=""
200 REM
210 REM NEUE GANS
220 REM
230 FOR D=0 TO 2:PUT SPRITE D,(0,0),4,7:NEXT D
240 SPRITE ON
250 STRIG(0) ON
260 ON SPRITE GOSUB 700
270 ON STRIG GOSUB 900
280 IF INKEY$<>"" THEN 280
290 DIVE=0:BULL=0
300 X=255:Y=181
310 T=T+1
320 V=RND(1)*6+4
330 H=RND(1)*100+20
400 REM
410 REM HAUPTSCHLEIFE
420 REM
430 PUT SPRITE 0,(X,H),14,N
440 Q$=INKEY$:IF Q$<>"" THEN W=ASC(Q$)*-(NOT HIT) ELSE W=0
450 IF W=28 THEN A=A+15
460 IF W=29 THEN A=A-15
470 PUT SPRITE 2,(A,184),11+10*HIT,5-HIT
480 X=X-V
490 N=N MOD 4+1
500 FOR E=0 TO 10
510 IF BULL THEN Y=Y-1.5:PUT SPRITE 1,(XA,Y),15,0 ELSE FOR D=
0 TO 2:NEXT D
```

```
520 NEXT E
530 IF Y<-4 THEN Y=181:BULL=0:STRIG(0) ON
540 IF ABS(INT(X)-A)<4 THEN SPRITE OFF:LINE(A+5,H+4)-(A+5,192
),15:LINE(A+5,H)-(A+5,192),4:SPRITE ON:IF RND(1)<.7+.35*HIT T
HEN IF HIT THEN 600 ELSE HIT=-1
550 IF X>0 AND NOT DIVE THEN 400 ELSE IF T=10 THEN 600 ELSE 2
00
600 REM
610 REM ENDE
620 REM
630 SCREEN0
640 PRINT "Gänse von ";G;" Abgeschoßsen ";T
650 END
700 REM
710 REM Gans getroffen
720 REM
730 SPRITE OFF
740 BEEP
750 PUT SPRITE 1,(0,0),4,7
760 D=H:E=1
770 G=G+1
780 SPRITE ON
790 X=X-V:D=D+E
800 PUT SPRITE 0,(X,D),1,D MOD 4+1
810 E=E+E/2
820 ON SPRITE GOSUB 1100
830 IF D<184 THEN FOR U=0 TO 100:NEXT U:GOTO 790
840 DIVE=-1
850 SOUND 6,10
860 SOUND 7,31
870 FOR D=250 TO 0 STEP -1:SOUND 10,D/20:NEXT D
880 IF HIT THEN HIT=0
890 RETURN
900 REM
910 REM Kugel abfeuern
920 REM
930 STRIG(0) OFF
940 SOUND 6,15
950 SOUND 7,31
960 SOUND 10,15
970 FOR D=0 TO 5:NEXT D
980 SOUND 10,0
990 BULL=-1
1000 XA=A
1010 RETURN
1100 REM
1110 REM Gans auf Geschütz abgestürzt
1120 REM
1130 SPRITE OFF:BEEP
1140 IF NOT HIT THEN HIT=-1
1150 RETURN
```

Stichwortverzeichis

A
ABS *43, 179*
Analytic Engine *130*
AND *187*
Anweisung *10*
Anweisungen, in Zeile *44*
Array *79*
Array-Element *80*
ASC *89, 179*
ASCII-Werte *89*
ATN *43, 179*
Ausgabe *10*
Ausgabegeräte *14*
AUTO *33, 154*

B
Babbage C. *130*
Bearbeitungszeichen *186*
BEEP *133, 154*
Befehl *10*
binäre Zahlen *56*
BLOAD *155*
BSAVE *155*
Byte *12*

C
CALL *155*
Cartridge *18*
CDBL *179*
CHR$ *90, 179*
CINT *179*
CIRCLE *119, 155*
CLEAR *155*
CLOAD *156*
CLOAD? *156*
CLOSE *149, 156*
CLS *63, 156*
Codes *151*
COLOR *115, 156*
Computer *9*
CONT *157*
COS *43, 179*
CRSLIN *180*
CSAVE *157*
CSGN *180*

D
D *55*
DATA *49, 157*
Database *28*
Dateien *148*
DBL *158*
DEF *158*
DEF FN *108, 157*
DEF USR *158*
Denken von Computer *95*
DELETE *35, 158*
DIM *79, 159*
doppelte Genauigkeit *55*
DRAW *119, 123, 159*

E
E *55*
Eingabe *10*
Eingabegerät *14*
ELSE *72, 162*
END *161*
ENIAC *28*
EOF *161, 180*
EQV *187*
ERASE *161*
ERL *161*
ERR *161*
ERROR *162*
EXP *54, 180*
Exponent *54*
Externer Speicher *85*

F
Fehlerbeseitigung *35*
Fehlermeldungen *188*
FIX *180*
Floppy Disk *13*
FOR...NEXT *67, 74, 162*
FRE *180*

G
Gänsespiel *101*
Genauigkeit *55*
GOSUB...RETURN *67, 104, 162*
GOTO *67, 68, 163*

H
Hardware *14*
Hexadezimalzahlen *57*
HEX$ *180*

I
IF...THEN *67, 69, 163*
IMP *187*
INKEY$ *92, 180*
INP *181*
INPUT *47, 163*
INPUT$ *164, 181*
INPUT# *164*
INSTR *181*
INSTR$ *92*
INT *43, 181*
Integer *56*
INTERVAL ON/OFF/STOP *164*

J
Joystick *116*

K
KEY *164*
KEY LIST *165*
KEY OFF *165*
KEY ON *165*
Klammern *22*

L
LEFT$ *88, 181*
LEN *87, 181*
LET *41, 165*
LINE *119, 123, 165*
LINE INPUT *150, 165*
LINE INPUT$ *165*
LIST *32, 166*
LLIST *166*
LOAD *166*
LOCATE *63, 165, 166*
LOG *43, 181*
LPOS *181*
LPRINT *166*
LPRINT USING *166*

M
Maschinenkodes *59*
Mantisse *54*
MAXFILE *150, 166*
MERGE *166*
Microprozessor *14*
MID$ *89, 167, 182*
Modem *14*
MOTOR ON/OFF *167*
MSX *17*
MSX-BASIC *9*
MSX-DOS *15*

N
Name, Variable *42*
NEW *32, 167*
NOT *187*

O
OCT$ *182*
Oktalzahlen *57*
ON ERROR GOTO *167*
ON...GOSUB *168*
ON...GOTO *67, 76, 167*
ON INTERVAL GOSUB *168*
ON KEY GOSUB *168*
ON SPRITE GOSUB *144, 168*
ON STOP GOSUB *168*
ON STRIG GOSUB *169*
OPEN *148, 169*
Operating System *15*
OR *187*
OUT *169*

P
PAD *182*
PAINT *125, 170*
PASCAL *19*
PASCAL B. *130*
PDL *182*
PEEK *182*
PLAY *134, 170, 182*
POINT *182*
POS *183*
Primzahlen *81*
PRINT *21, 61, 171*
PRINT# *173*
PRINT USING *63, 173*
PRINT # USING *173*
Prioritätsregeln *23*
Programm *10, 25*
Programmablaufplan *70*
Programmiersprache *14*
Prozessor *10*
PSET *113, 174*
PUT SPRITE *144, 174*

R
RAM *11*
READ *49, 174*
Relationszeichen *70*
REM *64, 174*
RENUM *33, 175*
RESTORE *51, 175*
RESUME *175*
RETURN *104*
RIGHT$ *88, 183*
RND *105, 183*
ROM *11*
RUN *31, 175*

S
SAVE *175*
SCREEN *111, 175*
SCREEN 2 *112*
SGN *43, 183*
SIN *43, 183*
Software *14*
SOUND *130, 176*
SPACE$ *91, 183*
SPC *183*
Speicheraufgliederung *153*
SPRITE *143*
SPRITE ON *144*
SPRITE ON/OFF/STOP *177*
SQR *43, 183*
Standardfunktionen *42*
STICK *183*
STOP *177*
STOP ON/OFF/STOP *177*
STR *157*
STR$ *90, 183*
STRIG *117, 184*
STRIG ON/OFF/STOP *177*
String *85*
STRING$ *91, 183*
String-Funktion *87*
String-Variable *85*
Subroutine *101*
SWAP *45, 177*

T
TAB *62, 184*
TAN *43, 184*
Teilen *22*
Text auf Bildschirm *148*
Trennungszeichen *44, 61*
TROFF *177*
TRON *177*

U
USING *63*
USR *178, 184*

V
VAL *90, 184*
Variable *39*
VARPTR *184*
Vesalius *131*
VPEEK *184*
VPOKE *178*

W
WAIT *178*
WIDTH *178*
Wissenschaftliche Schreibweise *53*

X
XOR *187*

Z
Z80 *11*
Zeilennummern *26*
zentrale Verarbeitungseinheit *10*

Was heißt eigentlich MSX? Weshalb sind gerade MSX-Computer dafür ausersehen, eine so wichtige Rolle zu spielen? Was kann man denn mit MSX-BASIC anfangen? In diesem leicht verständlichen Buch finden Sie die Antworten auf diese und ähnliche Fragen. Der Verfasser ging davon aus, daß der Leser ein Neuling in der Computerwelt ist. Systematisch, aber auf unterhaltsame Weise werden ihm die Geheimnisse des Programmierens in MSX-BASIC enthüllt und wird er in die Bedienung von Peripheriegeräten wie Datenrecorder, Drucker und Diskettenlaufwerke eingeführt.

PHILIPS

3122.285.70191